NUREG/CR-3134
SAND83-0074
RS
Printed April 1984

A SETS User's Manual for Vital Area Analysis

Desmond W. Stack, Mildred S. Hill

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

Prepared for
U. S. NUCLEAR REGULATORY COMMISSION

Available from

GPO Sales Program
Division of Technical Information and Document Control
U.S. Nuclear Regulatory Commission
Washington, D.C. 20555

and

National Technical Information Service
Springfield, Virginia 22161

ABSTRACT

This manual describes the use of the Set Equation Transformation System (SETS) for vital area analysis. Various techniques are presented for using SETS to solve vital area analysis fault trees. Depending on the input to SETS, the solution to the vital area analysis fault tree can be in terms of vital areas or primary events of the vital area analysis fault tree. The techniques presented are also suitable and efficient for other kinds of common cause analysis.

Contents

		Page
1.0	Introduction	1
2.0	The SETS Program	2
2.1	Introduction	2
2.2	Fault Tree Input	2
2.2.1	Example Fault Tree Input	4
2.2.2	Event Names	5
2.2.3	Primary Event Definitions	5
2.3	Intermediate (Gate) Event Definitions	6
3.0	SETS User Program	8
3.1	A Minimal Cut Set Algorithm	8
3.2	Understanding SETS User Programs	9
3.2.1	Boolean Equations	10
3.2.2	Procedure Calls	10
3.2.3	The Block File	12
3.2.4	The Equation File	12
3.3	Example SETS User Programs	14
3.3.1	Example 1. Minimal Cut Sets for the Top Event of the Figure 1 Fault Tree	14
3.3.2	Example 2. Minimal Cut Sets for the Top Event and Intermediate Events of a Fault Tree	16
4.0	SETS Input for Vital Area Analysis	20
4.1	The Sabotage Fault Tree	20
4.2	The Area Equations	20
4.3	SETS Input Example	21
4.3.1	The Fault Tree Input	21
4.3.2	The Area Equations	27
4.3.3	The OMEGA Block	29
5.0	The SETS User Program for Solving the Sabotage Fault Tree	30
5.1	Replicated Subtrees	30

Contents

		Page
5.2	AND Gates Above the Replicated Subtrees	31
5.3	Solving the Top Gate	31
5.4	SETS Program for Solving the Example Sabotage Fault Tree in Terms of Areas	32
5.4.1	The BLKSTAT and LDBLK Statements	33
5.4.2	Solving the Replicated Subtrees	33
5.4.3	Solving the AND Gates Above the Replicated Subtrees	34
5.4.4	Solving the Top Gate	34
5.4.5	Output of the SETS Program	34
5.5	Potential Problems Encountered with the SETS Program	37
5.5.1	A Replicated Subtree is too Large to Solve with a Single SUBINEQN and REDUCEQN Procedure	37
5.5.2	An AND Gate with More than Two Inputs	37
5.5.3	Using the FRMBLK Procedure in Large SETS User Programs	38
6.0	The Comp'ement Equation	41
6.1	Determining the Complement Equation Using SETS	41
6.2	An Example of Using SETS to Find the Complement Equation	41
6.3	A Potential Problem in Solving the Complement Equation	43
7.0	Minimal Cut Sets of Sabotage Acts	44
7.1	The SETS User Program for Identifying Scenarios	44
7.2	An Example of a SETS User Program for Identifying the Scenarios of Type 1 Vital Areas	45
7.2.1	The Scenario Equations	46
7.2.2	Setting the Type 2 Vital Areas to /OMEGA	47
7.2.3	The Equation File	48
7.2.4	Solving the Sabotage Fault Tree for Scenarios	48
7.2.5	The Solution Equation	49

Contents

Page

7.3 Potential Problems in Determining the Scenarios 50

7.3.1 Determining the Scenarios for a Single Type 1 Vital Area 57
7.3.2 Determining a Subset of Scenarios 61
7.3.3 Using Truncation when the STOP Option is Being Used 62

Appendix A. Procedures Available in SETS A-1

A.1 Read Block A-1

A.2 Read Fault Tree A-1

A.3 Print Equation A-2

A.4 Print Equation In Disjunctive Normal Form A-2

A.5 Delete Equation A-3

A.6 Substitute In Equation A-4

A.7 Reduce Equation A-5

A.8 Form Block A-7

A.9 Load Block A-8

A.10 Print Block A-9

A.11 Delete Block A-9

A.12 Block Status A-10

Appendix B. Execution Diagnostics B-1

B.1 SETS Errors B-1

B.2 SETS User Program Errors B-2

B.2.1 Special Fault Tree Error Messages B-3
B.2.2 Numbered Error Messages B-3

Appendix C. The Output of PRTBLK for the Example Sabotage Fault Tree and its Area Equations C-1

Figures

Figure 1. A Simple Fault Tree 2

Figure 2. Fault Tree Symbols 3

Figure 3. Processing Schematic for Procedure Calls 11

Figure 4. Another Simple Fault Tree 17

Figure 5. A Sabotage Fault Tree ..... Inside Back Cover

1.0 INTRODUCTION

This manual describes how to construct SETS user programs to perform a vital area analysis. SETS is a very general, flexible tool for manipulating Boolean equations. This manual, however, is designed specifically to describe the application of SETS to vital area analysis.

The SETS program is used to achieve three of the major objectives of a vital area analysis. These are:

- To identify the areas and combinations of areas in a nuclear power plant in which at least one collection of unauthorized acts can be accomplished which will lead to the release of significant amounts of radioactive material.

- To determine minimal sets of areas, the protection of which will interrupt all possible sequences of unauthorized acts which can result in radioactive release.

- To identify minimal collections of unauthorized acts which lead to radioactive release where all of the unauthorized acts in a given collection can be accomplished in the same area.

Chapters 2 and 3 contain an introduction to the SETS program. This material is condensed from A SETS User's Manual for the Fault Tree Analyst by R. B. Worrell and D. W. Stack, but does include all of the features of SETS necessary for vital area analysis. Chapter 4 describes the SETS input required for vital area analysis. In Chapter 5, the construction of a SETS user program to achieve the first objective is explained. In Chapter 6, the idea of a complement equation is introduced to satisfy the second objective. The third objective is treated in Chapter 7. Appendix A gives a concise summary of the procedures available in SETS which are used in a vital area analysis. Appendix B contains a list and explanation of all of the error messages which can be encountered in a vital area analysis.

An example vital area analysis fault tree is provided in Figure 7. This example is used in Chapters 4-7 to describe each step of the vital area analysis in detail.

2.0 THE SETS PROGRAM

2.1 Introduction

The Set Equation Transformation System (SETS) is a general tool for manipulating Boolean equations. The occurrence of the top event or any intermediate event of a fault tree can be represented by a Boolean equation. When this equation is transformed in a certain way, the fundamental ways that the top or intermediate event can occur (i.e., the minimal cut sets) may be read directly from the equation. Thus, SETS provides basic capabilities for manipulating Boolean equations to determine the minimal cut sets for the top event or any intermediate event of a fault tree.

The input to SETS consists of two parts: the fault tree description and the SETS user program. The input fault tree description employed in SETS is free format, and prepared from the graphical representation of a fault tree.

In addition to the input description of the fault tree, the analyst must prepare a SETS user program. This program translates the fault tree into a set of Boolean equations, and transforms these equations in a way which allows the cut sets to be obtained.

2.2 Fault Tree Input

The generation of the SETS computer input description of a fault tree usually proceeds from the familiar graphic representation of the tree such as the one shown in Figure 1.

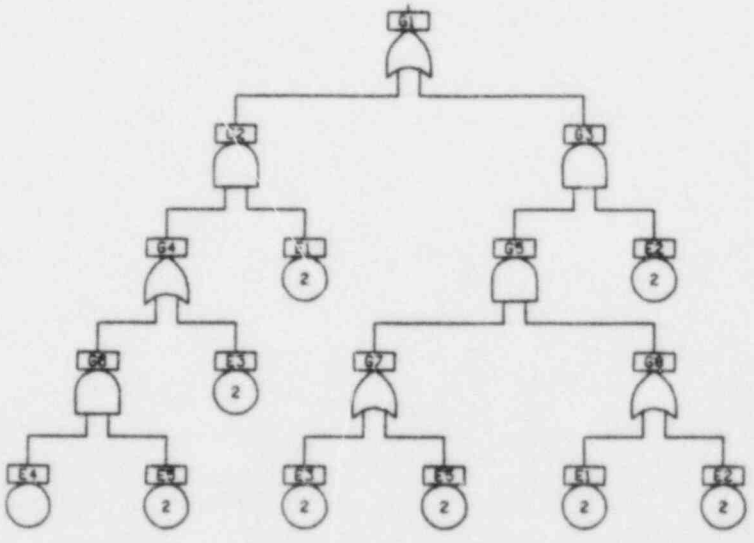


Figure 1. A Simple Fault Tree.

We assume that the reader is familiar with the basic fault tree terminology and symbols. The fault tree symbols used for sabotage fault trees are shown for convenient reference in Figure 2.

GATE SYMBOLS

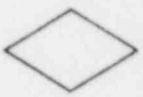
AND- Output fault occurs if all of the input faults occur

OR - Output fault occurs if at least one of the input faults occurs

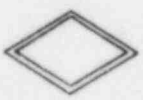
SPECIAL - Output fault occurs according to a logic function defined by the user

PRIMARY EVENT SYMBOLS

BASIC EVENT - An initiating fault requiring no further development

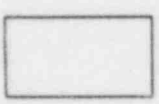


UNDEVELOPED EVENT - An event which is not further developed either because it is of insufficient consequence, or because information is unavailable



DEVELOPED EVENT - An event which could be further developed, or is developed elsewhere, but is treated here as a primary event

MISCELLANEOUS SYMBOLS



DESCRIPTION - Contains the description of an event

TRANSFER IN - Indicates that the tree is developed further at the occurrence of the corresponding TRANSFER OUT (e.g., on another page)

TRANSFER OUT - Indicates that this portion of the tree must be attached at the corresponding TRANSFER IN

Figure 2. Fault Tree Symbols.

2.2.1 Example Fault Tree Input

A listing of the data cards which would be used to input the fault tree shown in Figure 1 is as follows:

```
FAULT TREE$ FIG-1-FT.
     COMMENT$  INTERMEDIATE EVENT (GATE) DEFINITIONS.$
          OG$  G1.  IN$ G2, G3.
          AG$  G2.  IN$ G4, E1.  OUT$ G1.
          AG$  G3.  IN$ G5, E2.  OUT$ G1.
          OG$  G4.  IN$ G6, E3.  OUT$ G2.
          AG$  G5.  IN$ G7, G8.  OUT$ G3.
          AG$  G6.  IN$ E4, E5.  OUT$ G4.
          OG$  G7.  IN$ E5, E3.  OUT$ G5.
          OG$  G8.  IN$ E2, E1.  OUT$ G5.
     COMMENT$  PRIMARY EVENT DEFINITIONS.$
          BE$  E1.  OUT$ G2, G8.
          BE$  E2.  OUT$ G3, G8.
          BE$  E3.  OUT$ G4, G7.
          BE$  E4.  OUT$ G6.
          BE$  E5.  OUT$ G6, G7.
```

The fault tree input is written in a free format language. The representation of a fault tree which SETS can interpret is simply the string of characters that has been read from punched cards after all blank (space) characters have been purged from the string. This means that the SETS user need not be concerned about entering the input in certain columns or with certain spacing. As long as the delimiters (i.e., dollar signs, periods, and commas) are correctly placed, the input will be properly interpreted by SETS.

The first card in the fault tree input is the fault tree header which has the form:

FAULT TREE$ fault-tree-name.

where

"fault-tree-name" is an arbitrary name supplied by the analyst consisting of from 1 to 16 name symbols.

The delimiters "$" and "." are required.

The second card is a comment card which has the form:

COMMENT$ descriptive material $

where

"descriptive material" is supplied by the analyst.

Comments can occur throughout the input, but they must follow a "." delimiter. They do not contribute to the definition of the fault tree. The "$" delimiters are mandatory.

The rest of the cards are the intermediate and primary event definitions. The event definitions can occur in any order, but there must be exactly one definition for every event in the fault tree.

2.2.2 Event Names

As part of the event definitions, each primary and intermediate (gate) event must be assigned a name. Valid names consist of from 1 to 16 name symbols, where the name symbols are the capital letters A through Z, the digits 0 through 9 and the minus sign (used to represent a hyphen in a name). For example,

CH1-SQB, G4, 113, 53-A-GRND4

are all legitimate names, and, in fact, any ordering of from 1 to 16 name symbols is a name. Any name can be used as an event name in a fault tree with the exception of the name OMEGA which has special meaning.

2.2.3 Primary Event Definitions

The primary events of a fault tree are those events which, for one reason or another, have not been further developed. All primary events are treated identically during processing by SETS. The primary event definition for a basic event has the form:

BE$ basic-event-name. OUT$ output-event-list.

where

"BE" identifies the primary event as a basic event

"basic-event-name" is the name of the basic event supplied by the analyst

"output-event-list" is the list of gates to which the basic event has an output.

The other types of primary events have similar event definitions of the form:

UE$ undeveloped-event-name. OUT$ output-event-list.

DE$ developed-event-name. OUT$ output-event-list.

For example, the primary event definitions for the basic events E2 and E4 from the fault tree in Figure 1, are as follows:

BE$ E2. OUT$ G3, G8.

BE$ E4. OUT$ G6.

2.3 Intermediate (Gate) Event Definitions

The intermediate events of a fault tree are defined as logical combinations of other intermediate or primary events in the fault tree. Figure 2 describes some of the types of gates which are valid for use with SETS. They are:

1. The AND Gate.

2. The OR Gate.

3. The SPECIAL Gate.

The "SPECIAL" gate is different from the other gates because its logic function is defined by the user in the form of a Boolean expression. The SPECIAL gate makes it easy for the SETS user to describe, directly, such logic functions as the m-out-of-n gate instead of having to express it in terms of several AND and OR gates.

The intermediate event definition for an AND gate has the form:

AG$ and-gate-name. IN$ input-event-list. OUT$ output-event-list.

where

"AG" identifies the intermediate event as an AND gate

"and-gate-name" is the name of the AND gate supplied by the analyst

"input-event-list" is a list of gates and primary events which are inputs to the AND gate

"output-event-list" is a list of gates to which the AND gate has an output.

The top event of the tree has no output event list. The OR gate has a similar event definition of the form:

OG$ or-gate-name. IN$ input-event-list. OUT$ output-event-list.

For example, the intermediate event definitions for the OR gate G1 and the AND gate G2 from the fault tree in Figure 1 are as follows:

OG$ G1. IN$ G2, G3.
AG$ G2. IN$ G4, E1. OUT$ G1.

The intermediate event definition for the SPECIAL gate has a slightly different form:

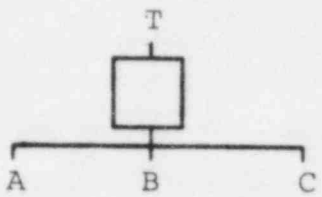
SG$ special-gate-name = logic-function. IN$ input-event-list.

OUT$ output-event-list.

where

"logic-function" is any properly formed Boolean expression.

The Boolean operations of conjunction (AND), disjunction (OR), and negation (NOT) that appear in the expression are represented by the symbols *, +, and /, respectively. The event names that appear in the Boolean expression must be the event names that appear in the input event list. Conversely, every event name in the input event list must appear in the Boolean expression. For example, the intermediate event definition for a SPECIAL gate which specifies that the output event, T, occurs when at least two of the three input events occur, can be represented in the following way:




```
SG$ T = A*B + B*C + A*C.
     IN$ A, B, C.
     OUT$ ....
```

3.0 SETS USER PROGRAM

As mentioned earlier, a fault tree may be represented by a collection of interrelated Boolean equations, one for each intermediate event. These equations can be processed by SETS to determine the minimal cut sets for any intermediate event in the fault tree. The fault tree analyst must direct this processing by writing a SETS user program, which is then read, interpreted, and executed by SETS. The SETS user programs will vary depending on such factors as the size and logical structure of the fault tree and the intermediate event for which the minimal cut sets are to be obtained. It is the ability to direct the processing which gives the SETS system its great generality and flexibility.

The names that occur in SETS user programs are constructed according to the same rules that were described for fault tree event names (Section 2.2.2).

3.1 A Minimal Cut Set Algorithm

A fault tree can be represented by a collection of interrelated Boolean equations, one for each intermediate event. These equations can be transformed to determine the minimal cut sets for any intermediate event in the fault tree. Three steps are necessary to find the minimal cut sets for a particular intermediate event:

Generate all of the intermediate event equations defined by the fault tree.

Generate an equation for the selected intermediate event as a function of only primary events by a repeated substitution process using the intermediate event equations generated in Step 1.

Reduce the equation resulting from Step 2 by applying the Boolean absorption identities P*P = P and P + P*Q = P.

The resulting equation, when printed in disjunctive normal (sum of products) form, is tantamount to a listing of the minimal cut sets since each Boolean product (AND) of primary events represents a minimal cut set.

Suppose that we want to use this minimal cut set algorithm to determine the minimal cut sets for the top event of the fault tree in Figure 1.

Step 1 -- Generate the intermediate event equations for the fault tree. To do this, simply write each intermediate gate event as a function of its input events:

G1 = G2 + G3

G2 = G4 * E1

G3 = G5 * E2

G4 = G6 + E3

G5 = G7 * G8

G6 = E4 * E5

G7 = E5 + E3

G8 = E2 + E1

Step 2 -- Generate an equation for G1 that is a function of only primary events. To do this, systematically eliminate each intermediate event on the right side of the equation for G1 by repeated substitution (i.e., replace each intermediate event by the right side of its equation from Step 1) until G1 is expressed entirely in terms of primary events.

G1 = G2 + G3

G1 = (G4 * E1) + (G5 * E2)

G1 = ((G6 + E3) * E1) + ((G7 * G8) *E2)

G1 = (((E4 *E5) + E3) * E1) + (((E5 + E3) * (E2 + E1)) * E2).

Step 3 -- Apply the identities P*P = P and P + P*Q = P to the equation generated in Step 2. The application of the identities can be easily seen by looking at a disjunctive normal form of the equation:

G1 = E4*E5*E1 + E3*E1 + E5*E2*E2 + E5*E1*E2 + E3*E2*E2 + E3*E1*E2.

The identity P*P = P, when applied to the 3rd and 5th terms of the equation will reduce them to E5*E2 and E3*E2, respectively. The 4th and 6th terms of the equation eliminated by the identity P + P*Q = P yielding the minimal cut sets for the top event of the fault tree:

G1 = E4*E5*E1 + E3*E1 + E5*E2 + E3*E2.

3.2 Understanding SETS User Programs

The minimal cut set algorithm described in the previous section is implemented for the fault tree in Figure 1 by the following very simple SETS user program:

```
PROGRAM$ EX1-MCS.
     RDFT (FIG-1-FT).
     LDBLK (FIG-1-FT).
     SUBINEQN (G1, G1-SUB).
     REDUCEQN (G1-SUB, G1-MCS).
     PRTEQNDNF (G1-MCS).
```

A SETS user program begins with a program header of the form:

PROGRAM$ program-name.

where

"program-name" is any name comprised of from 1 to 16 name symbols.

Following the program header are the program statements which are executed in the order that they occur. The first two statements in the program above, the RDFT and LDBLK statements, read and error check the fault tree input description and load its equivalent set of Boolean equations into the equation file. The third statement performs the substitution process which generates an equation for the top event G1 completely in terms of primary events. The last two statements perform the reduction process and print the final result in disjunctive normal form which is tantamount to a listing of the minimal cut sets.

Two kinds of statements can appear in a SETS user program: Boolean equations and procedure calls.

3.2.1 Boolean Equations

A Boolean equation defines an equivalence relationship between a Boolean variable on the left side of the equation and a Boolean expression on the right side of the equation. The Boolean variable on the left side of the equation is represented by a variable name comprised of 1 to 16 name symbols. The Boolean expression on the right can be a logical combination of variables involving the operations of conjunction (AND), disjunction (OR), and negation (NOT). A Boolean equation can be identified and referred to by its left side variable; "the equation for X" means the equation that has X as its left side variable.

3.2.2 Procedure Calls

A procedure call statement causes a predefined procedure to be executed. The following list summarizes the available procedure calls:

Procedures that process input		Read Block	RDBLK
		Read Fault Tree	RDFT
Procedures that process Boolean equations	Procedures that reference equations individually	Print Equation	PRTEQN
		Print Equation In Disjunctive Normal Form	PRTEQNDNF
		Delete Equation	DLTEQN
		Substitute In Equation	SUBINEQN
		Reduce Equation	REDUCEQN
	Procedures that reference blocks of equations	Print Block	PRTBLK
		Block Status	BLKSTAT
		Delete Block	DLTBLK
		Form Block	FRMBLK
		Load Block	LDBLK

From the above list, it can be seen that procedure calls process input, individual equations, or blocks of equations. To understand how the procedures are used, it is first necessary to understand how the block file and the equation file are used. The involvement of the block file and the equation file in the execution of the various procedures is depicted schematically in Figure 3. One or both of these files will be involved in the execution of every statement of a SETS user program, and the contents of the equation file and block file will vary as the execution of the SETS user program progresses.

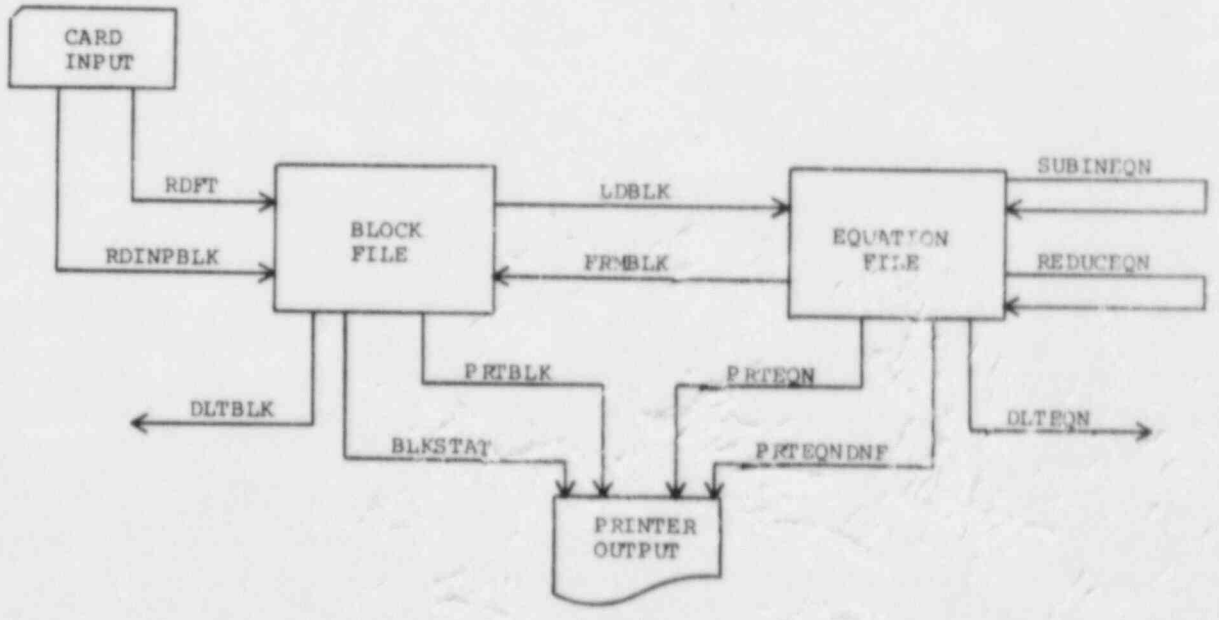


Figure 3. Processing Schematic for Procedure Calls.

3.2.3 The Block File

The block file is used to store groups of Boolean equations or blocks. As discussed earlier, a fault tree can be transformed into a set of Boolean equations, and is therefore a type of block. Each block is identified by a block name for easy reference. From Figure 3 it can be seen that SETS input is always entered in the block file using either the Read Fault Tree (RDFT) or Read Block (RDBLK) procedure. Once a block has been entered in the block file, the Load Block (LDBLK) procedure may be used to load its equations into the equation file for processing. In the case of a block that represents a fault tree, loading the block is usually aimed at determining the minimal cut sets. As shown in Figure 3, the other procedures which involve the block file are:

Delete Block (DLTBLK)
Print Block (PRTBLK)
Block Status (BLKSTAT)
Form Block (FRMBLK)

Delete Block deletes a block from the block file; Print Block prints the Boolean equations that constitute a block (additional information is printed for fault tree blocks); and Block Status lists the names of the blocks currently contained on the block file. Form Block is used to create a new block made up of equations from the equation file and to add that block to the block file. Using Form Block, it is thus possible to save in the block file equations that are created during the execution of one SETS user program for subsequent use in the same program, or for use in some other SETS user program at a later time.

3.2.4 The Equation File

In order to process the individual equations of a block, the block must be loaded into the equation file using the Load Block procedure call. In addition, equations may be entered in the equation file one at a time by using Boolean equation statements in a SETS user program. Figure 3 shows the procedures which reference the equation file. They are:

Print Equation (PRTEQN)
Print Equation In Disjunctive Normal Form (PRTEQNDNF)
Delete Equation (DLTEQN)
Substitute In Equation (SUBINEQN)
Reduce Equation (REDUCEQN)

Print Equation prints an equation in factored form; Print Equation In Disjunctive Normal Form prints an equation in sum of products form; and Delete Equation deletes an equation from the equation file. Substitute In Equation and Reduce Equation perform steps 2 and 3, respectively, of the minimal cut set algorithm that was described in section 3.1. Specifically, Substitute In Equation performs repeated substitutions on an equation until the right hand side of that equation consists entirely of primary events;

and Reduce Equation applies the Boolean absorption identities to an equation.

There is a fundamental difference in the way that the equation file and the block file are maintained. There can never be more than one equation with the same left side variable in the equation file, but it is possible to have more than one block with the same name in the block file. If the equation file already contains an equation for a particular variable when a new equation for that variable is defined (i.e., entered in the equation file), then the new equation will replace the existing equation. When a new block is defined, it is added to the block regardless of whether or not the block file already contains any blocks with the same name.

If more than one block with the same name ends up on the block file, the blocks can still be accessed individually since the blocks are loaded in the same order in which they appear on the block file. So if there are two blocks named X on the block file and we want the second one, the following SETS procedure calls will load the second block named X into the equation file:

```
LDBLK (X).

DLTEQN.

LDBLK (X).
```

The first LDBLK (X) statement loads the first block named X on the block file into the equation file. The DLTEQN procedure call deletes all of the equations in the equation file, so the equation file is now empty. The second LDBLK (X) loads the equations from the second block named X into the equation file. The equation file now contains all and only the equations from the second block named X.

It is generally a poor practice to have several blocks with the same name on the block file. If we have two blocks on the block file with the same name and we want to eliminate the first of these but keep the second one, the following SETS statements will accomplish this:

```
LDBLK (X).

DLTEQN.

LDBLK (X).

DLTBLK (X).

FRMBLK (X).
```

This example is similar to the previous example but the DLTBLK (X) and FRMBLK (X) statements have been added. The DLTBLK (X) statement drops all of the blocks named X from the block file.

The FRMBLK (X) statement forms a block named X of all of the equations in the equation file, which are the equations from the second block named X, and adds this block to the block file. Thus, there is now one and only one block named X on the block file.

Finally, suppose we have two blocks named X on the block file, but we want to keep the first block and drop the second one, then the SETS statements:

```
DLTEQN.

LDBLK (X).

DLTBLK (X).

FRMBLK (X).
```

will load the first block named X, delete all blocks named X from the block file, and form a block named X of the equations in the equation file, which are the equations from the first block named X. The first statement in the SETS segment, the DLTEQN statement, is used to make sure that the equation file is empty at the start of this segment.

Further discussion of the individual procedure calls appears in Appendix A. We will restrict this description of solving fault trees to techniques and procedures used for vital area analysis.

3.3 Example SETS User Programs

In this section we discuss two typical SETS user programs. These particular programs are applicable only to small fault trees.

3.3.1 Example 1. Minimal Cut Sets for the Top Event of the Figure 1 Fault Tree

Suppose that we want to write a SETS user program to determine the minimal cut sets for the top event of the fault tree in Figure 1 using the algorithm defined in Section 3.1. The SETS user program EX1-MCS, which is repeated here for convenience, accomplishes this task:

```
PROGRAM$ EX1-MCS.
     RDFT (FIG-1-FT).
     LDBLK (FIG-1-FT).
     SUBINEQN (G1, G1-SUB).
     REDUCEQN (G1-SUB, G1-MCS).
     PRTEQNDMF (G1-MCS).
```

The first two statements of the SETS user program EX1-MCS constitute an implementation of Step 1 of the minimal cut set algorithm. The first statement,

RDFT (FIG-1-FT)

is a call of the Read Fault Tree procedure. This statement is used to read the input description of the fault tree FIG-1-FT, and add to the block file a block which contains the intermediate event equations for the fault tree. The block has the same name as the fault tree.

The second statement,

LDBLK (FIG-1-FT)

is a call of the Load Block procedure. Execution of this statement causes the intermediate event equations contained in the block FIG-1-FT to be loaded (i.e., entered) in the equation file where they are available for further processing.

Statement 3 in the SETS user program EX1-MCS represent an implementation of Step 2 in the minimal cut set algorithm.

SUBINEQN (G1, G1-SUB)

is a call of the Substitute In Equation procedure. It is invoked to accomplish a repeated substitution process which begins with the right side of the equation specified by the first parameter in the call, G1. The substitutions continue using the equations that are currently in the equation file until no further substitutions can be made. Then, a new equation is defined and entered in the equation file. The left side variable of the new equation is the second parameter in the call, G1-SUB, and the right side is the expression that has been formed by the repeated substitution process.

The last two statements of the SETS user program EX1-MCS represent an implementation of Step 3 of the minimal cut set algorithm. The fourth statement,

REDUCEQN (G1-SUB, G1-MCS)

is a call of the Reduce Equation procedure which is used to apply the identities P*P = P and P + P*Q = P to the right side of the equation specified by the first parameter in the call, G1-SUB. Then, a new equation is defined with the second parameter in the call, G1-MCS, as its left side variable and the reduced expression as its right side. The last statement,

PRTEQNDNF (G1-MCS)

is a call of the Print Equation In Disjunctive Normal Form procedure which is used to print the equation for G1-MCS in a sum of products form. The product terms of this equation are the minimal cut sets for the top event of the fault tree:

Term Number	Number of Literals	
		G1-MCS =
1	2	E2 * E3 +
2	2	E1 * E3 +
3	2	E2 * E5 +
4	3	E1 * E4 * E5

Thus, the four minimal cut sets for G1 are: (E2, E3), (E1, E3), (E2, E5), and (E1, E4, E5).

The SETS user program EX1-MCS is an implementation of the algebraic algorithm for determining minimal cut sets. It shows what a SETS user program is like, and illustrates the use of several procedures. The general form of a procedure call statement is apparent from this example; a procedure call consists of a procedure identifier followed by a parameter list enclosed in parentheses. There are a few cases where parameters do not occur in the procedure call, but usually they are present.

3.3.2 Example 2. Minimal Cut Sets for the Top Event and Intermediate Events of a Fault Tree

Suppose we want to determine the minimal cut sets for the top event G1, and for the intermediate events G4 and G8, of the fault tree in Figure 4. Since G2 is a function of G4 and G8, and since the minimal cut sets for G4 and G8 are to be obtained anyway, the equations that represent the minimal cut sets for G4 and G8 can be determined first, and then used in the determination of the minimal cut sets for the top event, G1. This approach is implemented in the SETS user program:

```
PROGRAM$ EX2-MCS.
     RDFT (FIG-4-FT).
     LDBLK (FIG-4-FT).
     SUBINEQN (G4, G4).
     REDUCEQN (G4, G4).
     SUBINEQN (G8, G8).
     REDUCEQN (G8, G8).
     SUBINEQN (G1, G1).
     REDUCEQN (G1, G1).
     PRTEQN (G1, G4, G8).
     PRTEQNONF (G1, G4, G8).
```

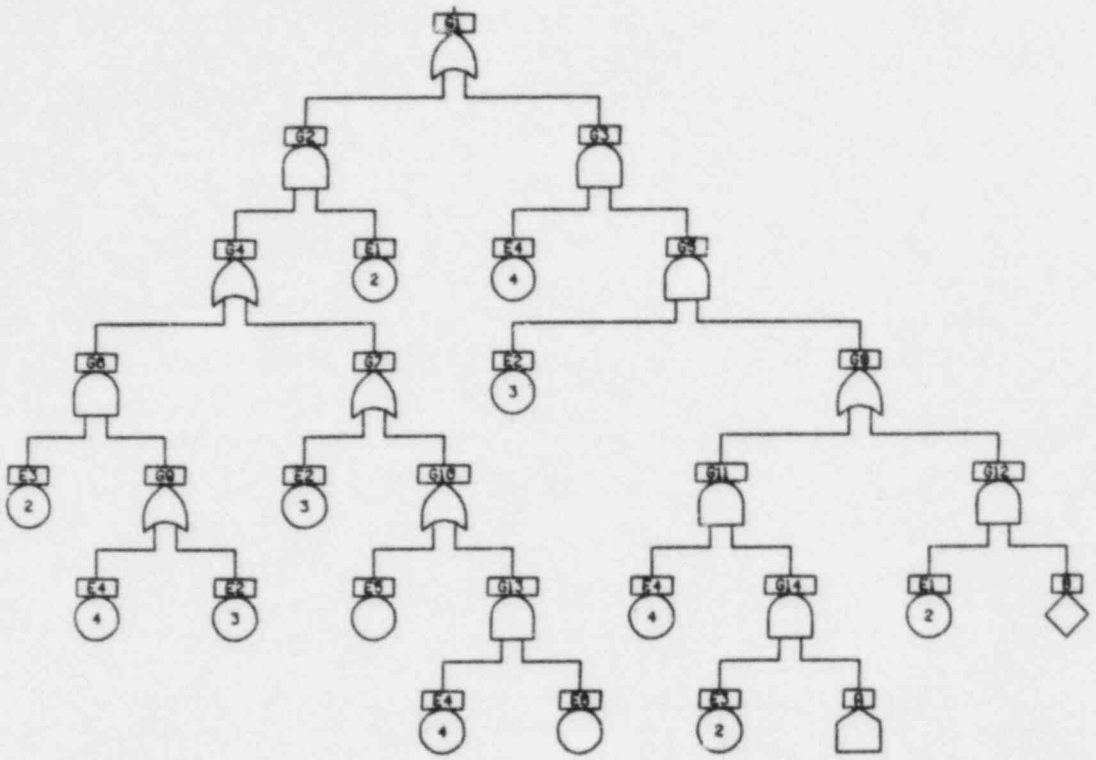


Figure 4. Another Simple Fault Tree.

As the SETS user program EX2-MCS is executed, the equations for G1, G4, and G8 in the equation file are redefined. The first two statements of the SETS user program,

```
RDFT (FIG-4-FT)
LDBLK (FIG-4-FT)
```

accomplish the processing of the fault tree and the loading of the intermediate event equations into the equation file. This establishes the original equation for G1, G4, and G8. The rest of the processing specified in the SETS user program for each of these three events is similar, and it will only be described for the intermediate event G4.

After the execution of the Load Block procedure, the equation for G4 in the equation file is:

```
G4 = G6 + G7
```

The third statement in the SETS user program,

```
SUBINEQN (G4, G4)
```

is a call of the Substitute In Equation procedure. Beginning with a copy of the right side of the equation specified by the first parameter, G4, the repeated substitution process is applied to form the right side of the new equation that will be defined and entered in the equation file by this procedure. The left side variable of the new equation is the second parameter in the procedure call, G4. Thus, a new equation for G4 is defined, and it replaces the old equation for G4 in the equation file. The equation for G4 that is in the equation file after the execution of this statement is:

```
G4 = ( (E3 * (E4 + E2) ) + (E2 + (E5 + (E4 * E6) ) ) )
      1 2     3       3 2   2     3     4       4 3 2 1
```

The fourth statement in the SETS user program,

```
REDUCEQN (G4, G4)
```

is a call of the Reduce Equation procedure which will once more define a new equation for G4. A copy of the right side expression of the equation specified by the first parameter, G4, is subjected to the application of the identities P*P = P and P + P*Q = P which results in the elimination of one term from the expression. A factored form of the remaining four terms becomes the right side of the new equation that will be defined and entered in the equation file by this procedure. The second parameter, G4, is the left side variable for the new equation. The new equation for G4,

```
G4 = E4 * (E6 + E3) + E5 + E2
          1       1
```

is a representation of the minimal cut sets for G4 and it replaces the old equation for G4 in the equation file.

The equations representing the minimal cut sets for G8 and G1 are developed in the same way that the equation representing the minimal cut sets for G4 is developed. The development of the equation for G1, however, makes use of the minimal cut set equations for G4 and G8 which were previously computed. The final two statements of the SETS user program,

```
PRTEQN (G1, G4, G8)
PRTEQNDNF (G1, G4, G8)
```

are used to print the minimal cut set equations for G1, G4, and G8. The equations are printed first in the factored form that they have in the equation file, and then in a disjunctive normal form which tantamount to a listing of the minimal cut sets for G1, G4, and G8:

G1 = E1 * (E4 * (E6 + E3) + E5 + E2) + E4 * E2 * E3 * A

G4 = E4 * (E6 + E3) + E5 + E2

G8 = E4 * E3 * A + E1 * B

Term Number	Number of Literals	G1 =
1	2	E1 * E5 +
2	2	E1 * E2 +
3	3	E1 * E4 * E6 +
4	3	E1 * E4 * E3 +
5	4	E4 * E2 * E3 * A

Term Number	Number of Literals	G4 =
1	1	E5 +
2	1	E2 +
3	2	E4 * E6 +
4	2	E4 * E3

Term Number	Number of Literals	G8 =
1	2	E1 * B +
2	3	E4 * E3 * A

There is an undeveloped event, B, which occurs in the equation for G8, but does not occur in the reduced equation for G1. This may signify that it is not necessary to develop the event B, because it is not involved in the minimal cut sets for the top event of the fault tree.

4.0 SETS INPUT FOR VITAL AREA ANALYSIS

Vital area analysis is the analytical procedure used to systematically identify the areas of a nuclear power plant that require physical protection. The inputs to the SETS portion of the analysis are the sabotage fault tree and a set of Boolean equations which identify the areas, or logical combinations of areas, in which each sabotage action depicted on the fault tree can occur.

4.1 The Sabotage Fault Tree

The top event of the sabotage fault tree is the release of significant amounts of radioactive material as a result of sabotage. The top event is developed into logical combinations of events until the development terminates in primary events. The primary events are sabotage actions, such as disabling a valve or cutting a pipe, which can lead to the undesired release of radioactive material. Each primary event can occur in some area or logical combination of areas in the plant.

The fault tree is read by SETS using the RDFT procedure. This procedure causes a block to be formed, with the name of the block being the same name as the fault tree, and adds the block to the block file. Both a linguistic representation and a Boolean representation of the fault tree are stored on the block. The Boolean representation is constructed by associating a Boolean equation with each gate in the fault tree. The left-hand-side of the Boolean equation is the name of the gate. The right-hand-side of the equation is a Boolean expression. The variables in the expression correspond to the names of the inputs to the gate. The variables are related by the appropriate Boolean operation depending on the type of gate.

4.2 The Area Equations

Each sabotage action represented by a primary event of the fault tree can occur in some area or some combination of areas. This area information is represented by a Boolean equation. For example, the equation:

```
B1 = AREA1 + AREA2 + (AREA3*AREA4) + AREA5
```

signifies that the sabotage act represented by B1 can be accomplished in any one of the areas 1, 2, or 5. Additionally, the sabotage act can take place in areas 3 and 4 but both areas must be accessed to accomplish the sabotage act. The RDBLK procedure is used to read the set of area equations, form a block containing the equations and add it to the block file.

If the fault tree is loaded into the equation file in SETS and a SUBINEQN is performed on any gate in the fault tree, the substitution process continues until it terminates on the primary

events in the fault tree. The minimal cut set algorithm will therefore produce minimal cut sets which are in terms of the primary events. However, if the fault tree and the Boolean equations which represent the area information for the primary events are loaded into the equation file, the substitution process continues through the primary events since there are equations for the primary events. In this case the substitution process terminates on the variables which represent the areas in which the sabotage acts can occur. Therefore, the minimal cut sets produced will be in terms of the areas in which the sabotage acts can occur. Thus, the minimal cut sets can be either in terms of sabotage acts or the areas in which these acts can occur, depending on what is entered into the equation file. Since vital area analysis is concerned with the areas of a nuclear power plant which require protection, we are interested in obtaining minimal cut sets in terms of areas.

4.3 SETS Input Example

The following SETS user program reads the fault tree and area equations, forms a block named SABOTAGE-FT for the fault tree and a block named AREA-EQUATIONS for the area equations and adds these blocks to the block file. These blocks are now ready to use in any subsequent SETS user program. There may be certain events in the fault tree which the analyst assumes will always occur, or will never occur. The last input block read in, OMEGA-ASSUMPTION, will later be used to implement these assumptions.

```
PROGRAM$EXAMPLE-1.
RDFT(SABOTAGE-FT).
PRTBLK(SABOTAGE-FT).
RDBLK(AREA-EQUATIONS).
PRTBLK(AREA-EQUATIONS).
RDBLK(OMEGA-ASSUMPTION).
PRTBLK(OMEGA-ASSUMPTION).
BLKSTAT.
```

4.3.1 The Fault Tree Input

The first SETS statement, RDFT (SABOTAGE-FT), reads the fault tree and forms a block named SABOTAGE-FT. The fault tree name must be the same as in the RDFT procedure call. The example fault tree has the following input:

```
RDFT (SABOTAGE-FT).

        FAULTTREE$SABOTAGE-FT.
        OG$  TOP.          IN$FM-TI,FM-ILOCA.
        OG$  FM-ILOCA.          IN$LI-MSD. OUT$TOP.
        AG$  LI-MSD.         IN$L-MSD,LI. OUT$FM-ILOCA.
        DE$  LI.              OUT$LI-MSD.
        BE$  LOSPW.           OUT$AC-4160-B1J-NP,AC-4160-B1J-AP,
                 AC-4160-B1H-NP,AC-4160-B1H-AP.
        DE$  RPS-D.           OUT$TMS-D.
        OG$  ADHRS-D.          OUT$DHRS-D. IN$SSRS-D,AFWS-LO-IHR.
        AG$  FM-TI.           IN$TI-RT,TMS-D. OUT$TOP.
        OG$  TI-RT.              IN$OI-RC,IHR-NHRS. OUT$FM-TI.
        OG$  TMS-D.            IN$RPS-D,DHRS-D. OUT$FM-TI.
        DE$  OI-RC.            OUT$ TI-RT.
        DE$  IHR-NHRS.            OUT$TI-RT.
        AG$  DHRS-D.           IN$ADHRS-D. OUT$TMS-D.
        DE$  SSRS-D.          OUT$ADHRS-D.
        OG$  AFWS-LO-IHR.  OUT$ADHRS-D.  IN$AFO-IHR.
        OG$ AFO-IHR.    IN$AFO-PMD-HS,AFO-PM-L,AFO-CS-PMS. OUT$AFWS-LO-IHR.
        DE$ AFO-PMD-HS.   OUT$AFO-IHR.
        AG$ AFO-PM-L.      IN$ AFO-MD-P1A,AFO-MD-P1B.
                 OUT$AFO-IHR.
        DE$ AFO-CS-PMS.  OUT$AFO-IHR.
        OG$ AFO-MD-P1A.  IN$AFO-P1A-D,AFO-P1A-EPW. OUT$AFO-PM-L.
        OG$ AFO-P1A-D.    IN$AFO-P1A-B,AFO-P1A-AUX,AFO-P1A-CSG.
                          OUT$AFO-MD-P1A.
        OG$ AFO-P1A-EPW.IN$AC-4160-B1H.
                    OUT$AFO-MD-P1A.
        OG$ AFO-P1A-AUX.IN$AFO-P1A-COOL.
                          OUT$ AFO-P1A-D.
        BE$ AFO-P1A-B.   OUT$AFO-P1A-D.
        BE$ AFO-P1A-CSG. OUT$AFO-P1A-D.
        DE$ AFO-P1A-COOL.OUT$AFO-P1A-AUX.
        OG$ AFO-MD-P1B.  IN$AFO-P1B-D,AFO-P1B-EPW. OUT$AFO-PM-L.
        OG$ AFO-P1B-D.    IN$AFO-P1B-B,AFO-P1B-AUX,AFO-P1B-CSG.
                          OUT$AFO-MD-P1B.
        OG$ AFO-P1B-EPW.IN$AC-4160-B1J.
                    OUT$AFO-MD-P1B.
        OG$ AFO-P1B-AUX.IN$AFO-P1B-COOL.
                          OUT$ AFO-P1B-D.
        BE$ AFO-P1B-B.   OUT$AFO-P1B-D.
        BE$ AFO-P1B-CSG. OUT$AFO-P1B-D.
        DE$ AFO-P1B-COOL.OUT$AFO-P1B-AUX.
        OG$  L-MSD.  IN$ECRS-D,ECIS-D,PAHRS-D.  OUT$LI-MSD.
        UE$  PAHRS-D.          OUT$L-MSD.
        AG$  ECRS-D.  OUT$ L-MSD.  IN$CS1A-IHR,CS1B-IHR.
        OG$ ECIS-D.  OUT$L-MSD. IN$CHO-IHR.
        OG$ CS1A-IHR.   IN$CS1A-PMD-HS,CS1A-HS,CS1A-PM-L,CS1A-HS-PMS.
                 OUT$ECRS-D.
        OG$ CS1A-PMD-HS. IN$CS1A-CX-SUMP-B,CS1A-PP-PMD-B.
                          OUT$CS1A-IHR.
        AG$ CS1A-PM-L.  IN$CS1A-MD-P1A.
```

```
                    OUT$ CS1A-IHR.
OG$ CS1A-HS-PMS.  IN$CS1A-PP-PMS-B.
OUT$CS1A-IHR.
DE$ CS1A-HS.   OUT$CS1A-IHR.
BE$ CS1A-PP-PMD-B. OUT$CS1A-PMD-HS.
BE$ CS1A-CX-SUMP-B.   OUT$CS1A-PMD-HS.
BE$ CS1A-PP-PMS-B. OUT$CS1A-HS-PMS.
OG$ CS1A-MD-P1A.  IN$CS1A-P1A-D,CS1A-P1A-EPW. OUT$CS1A-PM-L.
OG$ CS1A-P1A-D.   IN$CS1A-P1A-B,CS1A-P1A-CSG.
                   OUT$CS1A-MD-P1A.
OG$ CS1A-P1A-EPW.IN$AC-480V-B480H.
             OUT$CS1A-MD-P1A.
BE$ CS1A-P1A-B.   OUT$CS1A-P1A-D.
BE$ CS1A-P1A-CSG. OUT$CS1A-P1A-D.
OG$ CS1B-IHR.   IN$CS1B-PMD-HS,CS1B-HS,CS1B-PM-L,CS1B-HS-PMS.
          OUT$ECRS-D.
OG$ CS1B-PMD-HS. IN$CS1B-CX-SUMP-B,CS1B-PP-PMD-B.
                   OUT$CS1B-IHR.
AG$ CS1B-PM-L.  IN$CS1B-MD-P1B.
                  OUT$ CS1B-IHR.
OG$ CS1B-HS-PMS.  IN$CS1B-PP-PMS-B.
OUT$CS1B-IHR.
DE$ CS1B-HS.  OUT$CS1B-IHR.
BE$ CS1B-PP-PMD-B. OUT$CS1B-PMD-HS.
BE$ CS1B-CX-SUMP-B.   OUT$CS1B-PMD-HS.
BE$ CS1B-PP-PMS-B. OUT$CS1B-HS-PMS.
OG$ CS1B-MD-P1B.  IN$CS1B-P1B-D,CS1B-P1B-EPW. OUT$CS1B-PM-L.
OG$ CS1B-P1B-D.   IN$CS1B-P1B-B,CS1B-P1B-CSG.
                   OUT$CS1B-MD-P1B.
OG$ CS1B-P1B-EPW.IN$AC-480V-B480J.
             OUT$CS1B-MD-P1B.
BE$ CS1B-P1B-B.   OUT$CS1B-P1B-D.
BE$ CS1B-P1B-CSG. OUT$CS1B-P1B-D.
OG$ SWA-IHR.   IN$SWA-PMD-HS,SWA-CS-PMS,SWA-PM-L.
          OUT$CHO-P1A-COOL,CHO-P1B-COOL.
OG$ SWA-PMD-HS.IN$SWA-HX-OILCO-B,SWA-PP-PMD-B,SWA-VV-PMD.
          OUT$SWA-IHR.
OG$ SWA-PM-L.      IN$ SWA-MD-P10A,AC-4160-B1H.
          OUT$SWA-IHR.
DE$ SWA-CS-PMS.  OUT$SWA-IHR.
BE$ SWA-HX-OILCO-B.  OUT$SWA-PMD-HS.
BE$ SWA-PP-PMD-B.  OUT$SWA-PMD-HS.
BE$ SWA-VV-PMD.OUT$SWA-PMD-HS.
DE$ SWA-MD-P10A. OUT$SWA-PM-L.
OG$ SWB-IHR.   IN$SWB-PMD-HS,SWB-CS-PMS,SWB-PM-L.
          OUT$CHO-P1A-COOL,CHO-P1B-COOL.
OG$ SWB-PMD-HS.IN$SWB-HX-OILCO-B,SWB-PP-PMD-B,SWB-VV-PMD.
          OUT$SWB-IHR.
OG$ SWB-PM-L.      IN$ SWB-MD-P10B,AC-4160-B1J.
          OUT$SWB-IHR.
DE$ SWB-CS-PMS.  OUT$SWB-IHR.
BE$ SWB-HX-OILCO-B.  OUT$SWB-PMD-HS.
BE$ SWB-PP-PMD-B.  OUT$SWB-PMD-HS.
BE$ SWB-VV-PMD.OUT$SWB-PMD-HS.
DE$ SWB-MD-P10B. OUT$SWB-PM-L.
OG$ CHO-IHR.   IN$CHO-PMD-HS,CHO-CS-PMS,CHO-PM-L. OUT$ECIS-D.
OG$ CHO-PMD-HS.IN$CHO-HX-RVESS-B,CHO-VV-PMD,CHO-PP-PMD-B.
```

```
          OUT$CHO-IHR.
AG$ CHO-PM-L.      IN$ CHO-MD-P1A,CHO-MD-P1B.
          OUT$CHO-IHR.
DE$ CHO-CS-PMS.  OUT$CHO-IHR.
BE$ CHO-HX-RVESS-B.  OUT$CHO-PMD-HS.
BE$ CHO-PP-PMD-B.  OUT$CHO-PMD-HS.
OG$ CHO-VV-PMD.OUT$CHO-PMD-HS.
          IN$CHO-MV1286ABCOC,CHO-MV1867ABCC,CHO-MV1867BDCC.
BE$ CHO-MV1286ABCOC.OUT$CHO-VV-PMD.
BE$ CHO-MV1867ABCC. OUT$CHO-VV-PMD.
BE$ CHO-MV1867BDCC. OUT$CHO-VV-PMD.
OG$ CHO-MD-P1A.  IN$CHO-P1A-D,CHO-P1A-EPW. OUT$CHO-PM-L.
OG$ CHO-P1A-D.    IN$CHO-P1A-B,CHO-P1A-AUX,CHO-P1A-CSG.
                    OUT$CHO-MD-P1A.
OG$ CHO-P1A-EPW.IN$AC-4160-B1H.
             OUT$CHO-MD-P1A.
OG$ CHO-P1A-AUX.IN$CHO-P1A-COOL.
                   OUT$ CHO-P1A-D.
BE$ CHO-P1A-B.    OUT$CHO-P1A-D.
BE$ CHO-P1A-CSG. OUT$CHO-P1A-D.
AG$ CHO-P1A-COOL.OUT$CHO-P1A-AUX. IN$SWA-IHR,SWB-IHR.
OG$ CHO-MD-P1B.  IN$CHO-P1B-D,CHO-P1B-EPW. OUT$CHO-PM-L.
OG$ CHO-P1B-D.    IN$CHO-P1B-B,CHO-P1B-AUX,CHO-P1B-CSG.
                    OUT$CHO-MD-P1B.
OG$ CHO-P1B-EPW.IN$AC-4160-B1J.
             OUT$CHO-MD-P1B.
OG$ CHO-P1B-AUX.IN$CHO-P1B-COOL.
                   OUT$ CHO-P1B-D.
BE$ CHO-P1B-B.    OUT$CHO-P1B-D.
BE$ CHO-P1B-CSG. OUT$CHO-P1B-D.
AG$ CHO-P1B-COOL.OUT$CHO-P1B-AUX. IN$SWA-IHR,SWB-IHR.
OG$ AC-4160-B1J. IN$EP-BS-1J-D,AC-4160-B1J-PS.
 OUT$AC-480V-B480J-PS,CHO-P1B-EPW,AFO-P1B-EPW,SWB-PM-L.
BE$ EP-BS-1J-D.  OUT$AC-4160-B1J.
AG$ AC-4160-B1J-PS.    IN$AC-4160-B1J-SB,AC-4160-B1J-NP.    OUT$AC-4160-B1J.
OG$ AC-4160-B1J-SB.  IN$CB-C1J-O,DG-NO3-L.  OUT$AC-4160-B1J-PS.
BE$ CB-C1J-O.        OUT$AC-4160-B1J-SB.
AG$ AC-4160-B1J-NP.  IN$AC-4160-B1J-AP,LOSPW.    OUT$AC-4160-B1J-PS.
OG$ AC-4160-B1J-AP.  IN$COMP-MPWT-D,LOSPW.  OUT$AC-4160-B1J-NP.
UE$ COMP-MPWT-D.  OUT$AC-4160-B1J-AP,AC-4160-B1H-AP.
OG$ AC-4160-B1H. IN$EP-BS-1H-D,AC-4160-B1H-PS.
 OUT$AC-480V-B480H-PS,CHO-P1A-EPW,AFO-P1A-EPW,SWA-PM-L.
BE$ EP-BS-1H-D.  OUT$AC-4160-B1H.
AG$ AC-4160-B1H-PS.    IN$AC-4160-B1H-SB,AC-4160-B1H-NP.    OUT$AC-4160-B1H.
OG$ AC-4160-B1H-SB.  IN$CB-C1H-O,DG-NO1-L.  OUT$AC-4160-B1H-PS.
BE$ CB-C1H-O.        OUT$AC-4160-B1H-SB.
AG$ AC-4160-B1H-NP.  IN$AC-4160-B1H-AP,LOSPW.    OUT$AC-4160-B1H-PS.
OG$ AC-4160-B1H-AP.  IN$COMP-MPWT-D,LOSPW.  OUT$AC-4160-B1H-NP.
OG$ DG-NO3-L.  IN$DG-NO3-CSG,DG-NO3-B,DG-NO3-AUX.
          OUT$AC-4160-B1J-SB.
DE$ DG-NO3-AUX.     OUT$DG-NO3-L.
BE$ DG-NO3-B.    OUT$DG-NO3-L.
DE$ DG-NO3-CSG.     OUT$DG-NO3-L.
OG$ DG-NO1-L.  IN$DG-NO1-CSG,DG-NO1-B,DG-NO1-AUX. OUT$AC-4160-B1H-SB.
DE$ DG-NO1-AUX.     OUT$DG-NO1-L.
BE$ DG-NO1-B.    OUT$DG-NO1-L.
DE$ DG-NO1-CSG.     OUT$DG-NO1-L.
```

```
OG$ AC-480V-B480H.  IN$ID-BS-480H-D,AC-480V-B480H-PS.
          OUT$CS1A-P1A-EPW.
OG$ AC-480V-B480H-PS.  IN$ID-TR-SSXFM1H-D,AC-4160-B1H.  OUT$AC-480V-B480H.
BE$ ID-BS-480H-D.  OUT$AC-480V-B480H.
BE$ ID-TR-SSXFM1H-D.  OUT$AC-480V-B480H-PS.
OG$ AC-480V-B480J.  IN$ID-BS-480J-D,AC-480V-B480J-PS.
          OUT$CS1B-P1B-EPW.
OG$ AC-480V-B480J-PS.  IN$ID-TR-SSXFM1J-D,AC-4160-B1J.  OUT$AC-480V-B480J.
BE$ ID-BS-480J-D.  OUT$AC-480V-B480J.
BE$ ID-TR-SSXFM1J-D.  OUT$AC-480V-B480J-PS.

THE NEW EQUATION BLOCK SABOTAGE-FT
HAS BEEN ADDED TO THE BLOCK FILE
```

The block SABOTAGE-FT contains both a linguistic representation of the fault tree, similar to the input form, and a set of Boolean equations for the fault tree. Both of these representations are printed by the PRTBLK procedure. The output of the PRTBLK (SABOTAGE-FT) statement is given in Appendix C.

4.3.2 The Area Equations

The RDBLK (AREA-EQUATIONS) statement reads the input block containing the area equations, forms a block named AREA-EQUATIONS, and adds the block to the block file. The name of the input block must be the same as in the RDBLK statement. The input block for the example is:

```
RDBLK (AREA-EQUATIONS).

        BLOCK$AREA-EQUATIONS.
        IHR-NHRS                    =SFGRDB.
        RPS-D                   =SFGRDA.
        OI-RC                   =SFGRDB.
        SSRS-D                  =MCC1JCV + MCC1HCV.
        AFO-PMD-HS  =CR + MCC1JCV.
        AFO-CS-PMS  =SFGRDA.
        AFO-P1A-B  =MCC1HCV.
        AFO-P1A-CSG  =MCC1HCV.
        AFO-P1A-COOL  =DGRM2.
        AFO-P1B-B  =MCC1JCV.
        AFO-P1B-CSG  =MCC1JCV.
        AFO-P1B-COOL  =DGRM1 + ESGRM.
        EP-BS-1J-D              =ESGRM.
        CB-C1J-O                =ESGRM.
        EP-BS-1H-D              =ESGRM.
        CB-C1H-O                =ESGRM.
        DG-NO3-B                =DGRM2.
        DG-NO1-B                =DGRM1.
        ID-BS-480H-D            =WSGRM.
        ID-TR-SSXFM1H-D         =2AB.
        ID-BS-480J-D            =ESGRM.
        ID-TR-SSXFM1J-D         =3EQUIPRM.
        AFWS-LO-IHR       =CR+MCC1HCV*MCC1JCV+SFGRDA.
        CS1A-PP-PMD-B           =3EQUIPRM.
        CS1A-CX-SUMP-B          =CR.
        CS1A-PP-PMS-B           =CR + 2AB.
        CS1A-P1A-B              =PMPHSE.
        CS1A-P1A-CSG            =CR.
        CS1B-PP-PMD-B           =CR.
        CS1B-CX-SUMP-B          =PMPHSE.
        CS1B-PP-PMS-B           =CR + 2AB*2RB.
        CS1B-P1B-B              =CR.
        CS1B-P1B-CSG            =CR.
        SWA-HX-OILCO-B          =TBSMTVVPT.
        SWA-PP-PMD-B            =3EQUIPRM+ESGRM*WSGRM+2AB.
        SWA-VV-PMD              =3EQUIPRM+2AB.
        SWB-HX-OILCO-B          =CHPMCUB.
        SWB-PP-PMD-B            =3EQUIPRM+ESGRM*WSGRM+2AB.
        SWB-VV-PMD              =3EQUIPRM+2AB.
        CHO-HX-RVESS-B          =CR.
        CHO-PP-PMD-B            =CHPMCUB+2AB.
        CHO-MV1286ABCOC         =CR+MCC1HCV*MCC1JCV+CHPMCUB.
        CHO-MV1867ABCC          =CR+MCC1HCV*MCC1JCV+2AB.
        CHO-MV1867BDCC          =CR+MCC1JCV*MCC1HCV+2AB.
```

```
CHO-P1A-B          =CHPMCUB.
CHO-P1A-CSG        =CHPMCUB.
CHO-P1B-B          =ESGRM.
CHO-P1B-CSG        =CHPMCUB.
LI  = CHPMCUB + 2AB + ESGRM + CR + RC.
CS1A-HS = SFGRDA + MCC1HCV.
CS1B-HS = SFGRDA + MCC1JCV + WSGRM.
SWA-CS-PMS = TBSMTVVPT.
SWB-CS-PMS = TBSMTVVPT.
SWA-MD-P10A = CR + 3EQUIPRM.
SWB-MD-P10B = CR + 3EQUIPRM.
CHO-CS-PMS = RWST + 2RWST+CR + 2AB + CHPMCUB.
DG-NO3-AUX = DGRM2.
DG-NO1-AUX = DGRM1.
DG-NO3-CSG = CR + RLYRM + DGRM1*DGRM2.
DG-NO1-CSG = CR + RLYRM + DGRM1*DGRM2.
```

This block can also be printed by the PRTBLK procedure, if desired. The output of PRTBLK (AREA-EQUATIONS) statement is given in Appendix C.

4.3.3 The OMEGA Block

Any events that the analyst assumes will always occur are set to OMEGA. In the subsequent Boolean manipulation of equations, this event will be treated as 1. Similarly, events that are assumed to never occur are set to /OMEGA, and will be treated as a 0. The RDBLK (OMEGA-ASSUMPTION) reads this set of equations, forms a block named OMEGA-ASSUMPTION and adds it to the block file. The input block name must be the same as in the RDBLK procedure call. The OMEGA-ASSUMPTION block for the example is:

```
BLOCK$OMEGA-ASSUMPTION.
LOSPW                =OMEGA.
PAHRS-D              =/OMEGA.
COMP-MPWT-D          =/OMEGA.
```

Like the other blocks, this block can be printed by a PRTBLK statement, giving:

```
* * * * BLOCK SET EQUATIONS * * * *
        (OMEGA-ASSUMPTION)


LOSPW = OMEGA

PAHRS-D = /OMEGA

COMP-MPWT-D = /OMEGA
```

Finally, a BLKSTAT procedure call will print the names of the blocks on the block file. For the example, the output of the BLKSTAT statement is:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  SABOTAGE-FT
                                                  2.  AREA-EQUATIONS
                                                  3.  OMEGA-ASSUMPTION
```

5.0 THE SETS USER PROGRAM FOR SOLVING THE SABOTAGE FAULT TREE

Chapter 3 describes writing a simple SETS user program to solve a fault tree. If the sabotage fault tree is small, a single SUBINEQN and REDUCEQN procedure call for the top gate will determine the minimal cut sets of the fault tree. However, most sabotage fault trees are much too large to be solved using this approach. Instead, various subtrees of the fault tree are solved until a solution for the top gate can be determined. Two kinds of subtrees are singled out for solution prior to solving the top gate; the replicated subtrees and the subtrees whose top gates are AND gates which are above the replicated subtrees in the fault tree. A replicated subtree is one whose top gate is a multiple output gate. These subtrees are readily identified on the fault tree plot since their top gates are plotted with transfer-out symbols as their outputs. The general steps involved in the SETS user program are:

- Load the fault tree, the area equations and the OMEGA assumptions into the equation file using the LDBLK procedure.

- Solve the replicated subtrees of the fault tree using the SUBINEQN and REDUCEQN procedures.

- Solve the AND gates of the fault tree which are located above the replicated subtrees using the SUBINEQN and REDUCEQN procedures.

- Solve the top gate of the fault tree using the SUBINEQN and REDUCEQN procedures.

- Form a block of the solution equation using the FRMBLK procedure.

- Print the solution using the PRTEQNDNF procedure.

We now discuss these steps in more detail. It helps to keep in mind what is in the equation file at all times since this is the key to understanding what is happening during the execution of a SETS user program.

5.1 Replicated Subtrees

The fault tree equations and the area equations are loaded into the equation file. Every gate in the fault tree has an equation in terms of its immediate inputs. The objective is to replace certain gate equations with their minimal cut set equations in the equation file. Then when these gates are encountered during the substitution process, their minimal cut set expressions will be encountered in the equation file rather than the much larger expressions that would be encountered if the substitution process continued with their immediate inputs. The

first group of gates chosen for solution is the collection of gates which are the top gates of the replicated subtrees. Since the top gate of such a subtree has multiple outputs, the subtree, in affect, appears in several places in the fault tree (in as many places as there are outputs for its top gate). If this subtree is replaced by a simpler representation, in this case its minimal cut set representation, there will be a savings every time it is encountered when substituting for the top gate of the fault tree. The order in which the replicated subtrees are solved is extremely important to the efficiency of the program. When solving a replicated subtree, if the substitution process encounters any top gates of replicated subtrees, their simpler representation will replace them in the equation being formed. These considerations give the following rule to be followed when determining the order in which the replicated subtrees are to be solved: A replicated subtree is solved only if all of the replicated subtrees which it contains have already been solved.

There is always some group of replicated subtrees which do not contain any replicated subtrees since a fault tree cannot have any cycles. These subtrees are solved first, in any order. They can be identified on the fault tree plot since they are plotted from their tops down to only primary events without any intervening transfer-in symbols. The remaining replicated subtrees are solved in the order given by following the above rule. The fault tree plot can be helpful when following the rule. The top gate of a replicated subtree is plotted down to primary events and transfer-in symbols, which represent inputs from replicated subtrees. If every replicated subtree represented by a transfer-in has already been solved, the replicated subtree can be solved.

5.2 AND Gates Above the Replicated Subtrees

Once all of the replicated subtrees have been solved, the second group of subtrees chosen for solution is the collection of subtrees which have AND gates located above the replicated subtrees as their top gates. The reason that AND gates are selected is that the maximum number of minimal cut sets for an AND gate is the product of the number of minimal cut sets for each of its inputs while the maximum number of minimal cut sets for an OR gate is the sum of the number of minimal cut sets for each of its inputs. Since some of the top events of replicated subtrees may have a large number of minimal cut sets, the distinction is an important one. The rule for determining the order in which to solve the AND gates above the replicated subtrees is similar to the rule for replicated subtrees: An AND gate is solved only if all of the AND gates below it (but above the replicated subtrees) have been solved. There may be single input AND gates in the fault tree which are merely for descriptive purposes. These gates are ignored when solving the fault tree.

5.3 Solving the Top Gate

Once all of the replicated subtrees and the AND gates above the replicated subtrees have been solved, we are ready to solve

the top gate of the fault tree. The SETS user program to solve the top gate:

```
SUBINEQN(TOP, TOP).

REDUCEQN(TOP, TOP).
```

is identical to the user program to solve the fault tree in a single step. However, the difference is that the equation file contains reduced, minimal cut set equations for the top gates of the replicated subtrees and the selected AND gates, so the substitution for the top gate brings in their minimal cut sets. This greatly increases the efficiency of the SETS execution of the user program.

The resulting minimal cut set equation is saved by the FRMBLK procedure and printed out using the PRTEQNDNF procedure. Since the area equations are in the equation file, the minimal cuts sets are in terms of areas. Each minimal cut set represents a set of areas in which at least one collection of sabotage acts, each of which can be accomplished in at least one area in the minimal cut set, is sufficient to cause the top event of the sabotage fault tree, i.e., the release of significant amounts of radioactive material. A minimal cut set of order one identifies a single vital area, called a Type 1 vital area. The areas represented in minimal cut sets of order greater than one are called Type 2 vital areas.

5.4 SETS Program for Solving the Example Sabotage Fault Tree in Terms of Areas

The following SETS user program determines the minimal cut sets of the sabotage fault tree, SABOTAGE-FT, in terms of areas. Each step in the program is discussed in detail following the program listing.

```
PROGRAM$EXAMPLE-2.
BLKSTAT.
LDBLK(SABOTAGE-FT,AREA-EQUATIONS,OMEGA-ASSUMPTION).
SUBINEQN(AC-4160-B1J,AC-4160-B1J).
REDUCEQN(AC-4160-B1J,AC-4160-B1J).
SUBINEQN(AC-4160-B1H,AC-4160-B1H).
REDUCEQN(AC-4160-B1H,AC-4160-B1H).
SUBINEQN(SWA-IHR,SWA-IHR).
REDUCEQN(SWA-IHR,SWA-IHR).
SUBINEQN(SWB-IHR,SWB-IHR).
REDUCEQN(SWB-IHR,SWB-IHR).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL,CHO-P1A-COOL).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL,CHO-P1B-COOL).
SUBINEQN(ECRS-D,ECRS-D).
REDUCEQN(ECRS-D,ECRS-D).
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L,AFO-PM-L).
```

```
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L,CHO-PM-L).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI,FM-TI).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD,LI-MSD).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP,VITAL-AREA-MCS).
FRMBLK(VITAL-AREA-MCS*ONLY$VITAL-AREA-MCS).
BLKSTAT.
PRTEQNDNF(VITAL-AREA-MCS).
```

5.4.1 The BLKSTAT and LDBLK Statements

The BLKSTAT statement is used to check the contents of the block file. For the example, the output of the BLKSTAT statement is:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  SABOTAGE-FT
                                                  2.  AREA-EQUATIONS
                                                  3.  OMEGA-ASSUMPTION
```

The LDBLK (SABOTAGE-FT, AREA-EQUATIONS, OMEGA-ASSUMPTION) load the fault tree equations, area equations and OMEGA equations into the equation file. The order in which the equations are loaded into the equation file is important, since there can never be more than one equation in the equation file with the same left-hand-side. So if event LOSPW had both an area equation and an OMEGA equation, the equation loaded last, in this case the OMEGA equation, is the one in the equation file.

5.4.2 Solving the Replicated Subtrees

By looking at the plot in Figure 5, we see the following gates are plotted with transfer-out symbols: AC-4160-B1J, AC-4160-B1H, SWA-IHR and SWB-IHR. These gates are top events of replicated subtrees. There are no transfer in symbols plotted in the subtrees for AC-4160-B1J and AC-4160-B1H, so these subtrees are solved first using the SUBINEQN and REDUCEQN procedures. There are no other replicated subtrees which do not have any transfer-in symbols plotted in their development, i.e., do not contain any replicated subtrees.

The replicated subtrees with top gates SWA-IHR and SWB-IHR both contain replicated subtrees. The subtree with SWA-IHR as its top event has a transfer-in from AC-4160-B1H, which is the top of a replicated subtree, but this subtree has already been solved so we can solve SWA-IHR with SUBINEQN and REDUCEQN procedure calls. Similarly, the subtree with top event SWB-IHR contains the replicated subtree with top event AC-4160-B1J, but this gate has also been solved so we can solve the subtree with top event SWB-IHR by a call of the SUBINEQN and REDUCEQN procedures.

At this point all of the replicated subtrees have been solved, so we can proceed to solving the AND gates located above the replicated subtrees.

5.4.3 Solving the AND Gates Above the Replicated Subtrees

The next step is to solve all of the AND gates (except single input AND gates) which are above the replicated subtrees. By looking at the plot, we can identify these gates as: CHO-P1A-COOL, CHO-P1B-COOL, CHO-PM-L, ECRS-D, LI-MSD, AFO-PM-L and FM-TI. The first AND gates to solve are those which do not have any other AND gates below them but above the replicated subtrees (since the replicated subtrees have been solved, we are not interested in solving any gates which they contain). The gates CHO-P1A-COOL, CHO-P1B-COOL, ECRS-D and AFO-PM-L all meet this criteria so they are solved using the SUBINEQN and REDUCEQN procedures. Next, we examine the fault tree for AND gates which do have AND gates below them, but all of the AND gates below them have been solved. The AND gate LI-MSD, for example, cannot yet be solved since AND gate CHO-PM-L below it and has not yet been solved. Gate CHO-PM-L, however, does meet this criteria since the only AND gates below it are CHO-P1A-COOL and CHO-P1B-COOL and these gates have been solved. Gates ECRS-D and FM-TI also meet this criteria since the AND gates below it are solved so gates CHO-PM-L, ECRS-D and FM-TI are solved using the SUBINEQN and REDUCEQN procedures. The last AND gate, LI-MSD, now meets the criteria for being solved so it is solved using the SUBINEQN and REDUCEQN procedures.

5.4.4 Solving the Top Gate

All of the replicated subtrees and the AND gates above the replicated subtrees have been solved, so we are now ready to solve the top gate. The top gate is also solved using the SUBINEQN and REDUCEQN procedures. The second parameter of the REDUCEQN call, VITAL-AREA-MCS, is used to name the final minimal cut set equation VITAL-AREA-MCS, or any other appropriate name may be used. This equation is stored on a block of the same name by the FRMBLK (VITAL-AREA-MCS *ONLY$ VITAL-AREA-MCS) statement. The ONLY option is used so that only the equation VITAL-AREA-MCS is stored on the block. Otherwise a FRMBLK procedure call forms a block of all the equations in the equation file.

5.4.5 Output of the SETS Program

The BLKSTAT statement is used to check the status of the block file at the end of the run. For the example, the BLKSTAT procedure call has the following output:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  SABOTAGE-FT
                                                  2.  AREA-EQUATIONS
                                                  3.  OMEGA-ASSUMPTION
                                                  4.  VITAL-AREA-MCS
```

The final statement in the program prints the vital area minimal cut sets. For the example, the printout is:

* * * * LITERAL OCCURRENCE TABLE * * * *

LITERAL	NUMBER OF OCCURRENCES
SFGRDB	3
SFGRDA	2
MCC1JCV	3
MCC1HCV	3
CR	1
DGRM2	2
DGRM1	2
ESGRM	1
WSGRM	1
2AB	1
3EQUIPRM	1
PMPHSE	1
TBSMTVVPT	1
CHPMCUB	1
RC	12
RWST	1
2RWST	1
RLYRM	1

THERE ARE 18 DIFFERENT LITERALS IN THE EQUATION FOR VITAL-AREA-MCS

TERM NUMBER	NUMBER OF LITERALS	
		VITAL-AREA-MCS =
1	1	CHPMCUB +
2	1	2AB +
3	1	ESGRM +
4	1	CR +
5	2	RC * RLYRM +
6	2	RC * 2RWST +
7	2	RC * RWST +
8	2	TBSMTVVPT * RC +
9	2	PMPHSE * RC +
10	2	3EQUIPRM * RC +
11	2	WSGRM * RC +
12	2	SFGRDA * RC +
13	2	SFGRDB * MCC1HCV +
14	2	SFGRDB * MCC1JCV +
15	2	SFGRDB * SFGRDA +
16	3	DGRM2 * DGRM1 * RC +
17	3	MCC1HCV * DGRM2 * RC +
18	3	MCC1JCV * DGRM1 * RC +
19	3	MCC1JCV * MCC1HCV * RC

Thus, there are 18 vital areas for the example. The first four, CHPMCUB, 2AB, ESGRM and CR are Type 1 vital areas. The remaining 14 vital areas are Type 2 vital areas. In order for a sabotage attempt to be successful, one must gain access to any one of the Type 1 vital areas or all of the Type 2 vital areas in one of the remaining 15 minimal cut sets.

5.5 Potential Problems Encountered with the SETS Program

If the sabotage fault tree is very large or complex, there are some problems that may be encountered when running the SETS user program. The most common of these, and possible solutions, are discussed below.

5.5.1 A Replicated Subtree is too Large to Solve with a Single SUBINEQN and REDUCEQN Procedure

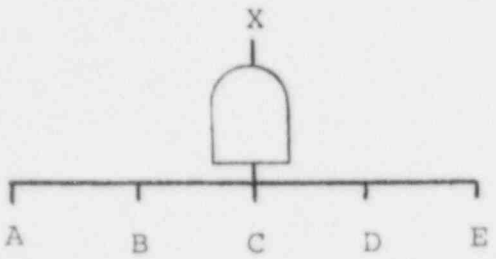
There may be a replicated subtree that is too large to solve in a reasonable amount of computer time and/or computer storage. First, check to make sure that the rule is being correctly followed so that all replicated subtrees contained in the problem replicated subtree have been solved. If this is the case, then the cause is that some gate in the problem replicated subtree which is above the solved replicated subtrees is generating a very large number of cut sets during substitution. For reasons previously described, this gate is usually an AND gate. By first solving this gate and replacing its cut sets by minimal cut sets, the replicated subtree can usually then be solved. If not, it may be necessary to solve all of the AND gates in the problem replicated subtree which are above the replicated subtrees contained in the problem replicated subtree. The order in which the AND gates are solved is the same as described earlier: an AND gate is solved only if all of the AND gates below it but above the replicated subtrees contained in the problem replicated subtree have been solved. These AND gates are solved using the SUBINEQN and REDUCEQN procedures. Once all of these AND gates are solved the problem replicated subtree is solved using the SUBINEQN and REDUCEQN procedures.

5.5.2 An AND Gate with More Than Two Inputs

There may be an AND gate in the sabotage fault tree which has more than two inputs and cannot readily be solved using the SUBINEQN and REDUCEQN procedures. Suppose an AND gate has five inputs, and each of these inputs has 100 minimal cut sets. Then the SUBINEQN and REDUCEQN procedure calls for this gate will generate $100^5 = 10^{10}$ or 10 billion cut sets. Since it is often not possible in terms of computer storage and/or computer time to reduce such an expression to minimal cut sets, we need a more efficient way of dealing with such gates. The general idea is to let some but not all of the minimal cut set expressions for the inputs to the AND gate be brought into the equation for the AND gate. All of the inputs to the AND gate should be solved

using the SUBINEQN and REDUCEQN procedures before attempting to solve the AND gate.

For example, suppose we wish to solve the following AND gate:



First, solve gates A, B, C, D, and E using the SUBINEQN and REDUCEQN procedures. Next, choose two of the five inputs to be brought in for the equation for X. If there is no good reason to choose a particular two inputs, then any two will do. However, often we know that some of the inputs do have something in common; for example, some of the inputs may share some of the same support systems. Inputs that have events in common will generally provide a greater reduction in the size of the equation being formed if they are brought in together. To bring in just A and B in the equation for X we use the STOP option for SUBINEQN. The SETS statement SUBINEQN (X, X *STOP$ C, D, E) lets the equations for A and B be brought in for X but stops the substitution process on inputs C, D, and E. The REDUCEQN procedure call REDUCEQN (X, X) will then determine the minimal cut sets of X with C, D, and E left in the equation for X so there will be a C*D*E expression in every minimal cut set for X. Continuing in this manner, we let the equations for C, D, and E be brought into the equation for X one at a time, unless there is some reason, such as shared support systems or events, we should bring in more than one at a time. For the example AND gate X the SETS user program to solve X is given by:

```
SUBINEQN (X, X *STOP$ C, D, E).
REDUCEQN (X, X).
SUBINEQN (X, X *STOP$ D, E).
REDUCEQN (X, X).
SUBINEQN (X, X *STOP$ E).
REDUCEQN (X, X).
SUBINEQN (X, X).
REDUCEQN (X, X).
```

5.5.3 Using the FRMBLK Procedure in Large SETS User Programs

Due to the problems discussed in the previous sections and other unanticipated errors in the SETS user program (see Appendix B), the SETS user program may not execute to completion. If this occurs, then all of the gates of the fault tree which have been

solved up to the point of the abnormal termination of the SETS program will have to be solved again unless steps have been taken in the SETS user program to save the minimal cut sets of the solved gates.

Each time a minimal cut set equation is determined, it can be saved using a FRMBLK procedure call. The most convenient way of doing this is to use a FRMBLK(A) or FRMBLK(STEPA) statement without using the ONLY option to form the first such block. Then subsequent blocks are formed after each REDUCEQN (or whenever the analyst chooses). These blocks are usually named something similar like B, C, D, ... or STEPB, STEPC, STEPD, ... If the SETS program has an abnormal termination, then the last block formed contains all of the reduced equations determined up to the point the block was formed. To restart the run, after the problem has been corrected, the last block formed is loaded into the equation file. Since the equation file now contains everything that was in it when the program aborted, it is important that no other blocks be loaded, such as the fault tree equations, since these equations will replace the solved equations in the equation file.

Since these blocks serve no other useful propose than allowing us to restart the program without having lost any information, they are deleted, using the DLTBLK procedure, after the vital area minimal cut sets have been determined.

As an example of this approach, we can modify the SETS user program for the example (Section 5.4) as follows:

```
PROGRAM$EXAMPLE-3.
BLKSTAT.
LOBLK(SABOTAGE-FT,AREA-EQUATIONS,OMEGA-ASSUMPTION).
SUBINEQN(AC-4160-B1J,AC-4160-B1J).
REDUCEQN(AC-4160-B1J,AC-4160-B1J).
FRMBLK(STEPA).
SUBINEQN(AC-4160-B1H,AC-4160-B1H).
REDUCEQN(AC-4160-B1H,AC-4160-B1H).
FRMBLK(STEPB).
SUBINEQN(SWA-IHR,SWA-IHR).
REDUCEQN(SWA-IHR,SWA-IHR).
FRMBLK(STEPC).
SUBINEQN(SWB-IHR,SWB-IHR).
REDUCEQN(SWB-IHR,SWB-IHR).
FRMBLK(STEPD).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL,CHO-P1A-COOL).
FRMBLK(STEPE).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL,CHO-P1B-COOL).
FRMBLK(STEPF).
SUBINEQN(ECRS-D,ECRS-D).
REDUCEQN(ECRS-D,ECRS-D).
FRMBLK(STEPG).
```

```
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L,AFO-PM-L).
FRMBLK(STEPH).
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L,CHO-PM-L).
FRMBLK(STEPI).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI,FM-TI).
FRMBLK(STEPJ).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD,LI-MSD).
FRMBLK(STEPK).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP,VITAL-AREA-MCS).
FRMBLK(VITAL-AREA-MCS*ONLY$VITAL-AREA-MCS).
OLTBLK(STEPA,STEPB,STEPC,STEPD,STEPE,STEPF,STEPG,STEPH,STEPI,STEPJ,STEPK).
BLKSTAT.
PRTEQNONF(VITAL-AREA-MCS).
```

If the program aborted after block STEPD was formed, say solving gate CHO-P1A-COOL, then the next SETS user program would be:

```
PROGRAM$EXAMPLE-4.
OLTBLK(STEPA,STEPB,STEPC).
BLKSTAT.
LDBLK(STEPD).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL,CHO-P1A-COOL).
FRMBLK(STEPE).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL,CHO-P1B-COOL).
FRMBLK(STEPF).
SUBINEQN(ECRS-0,ECRS-0).
REDUCEQN(ECRS-0,ECRS-0).
FRMBLK(STEPG).
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L,AFO-PM-L).
FRMBLK(STEPH).
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L,CHO-PM-L).
FRMBLK(STEPI).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI,FM-TI).
FRMBLK(STEPJ).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD,LI-MSD).
FRMBLK(STEPK).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP,VITAL-AREA-MCS).
FRMBLK(VITAL-AREA-MCS*ONLY$VITAL-AREA-MCS).
OLTBLK(STEPE,STEPF,STEPG,STEPH,STEPI,STEPJ,STEPK).
BLKSTAT.
PRTEQNONF(VITAL-AREA-MCS).
```

6.0 THE COMPLEMENT EQUATION

If one gains access to all of the vital areas in any minimal cut set, then one can cause the top event of the sabotage fault tree to occur by some collection of sabotage acts. Conversely, if we can deny access to at least one vital area in every minimal cut set, then the top event cannot occur. One way to determine such a group of vital areas to deny access to is to form the complement of the minimal cut set equation and then determine the minimal cut sets of the complement equation.

6.1 Determining the Complement Equation Using SETS

If the minimal cut set equation has TOP as the Boolean variable on its left-hand-side, then the following SETS user program identifies a complement expression for TOP by a SUBINEQN and determines the minimal cut sets of TOP by a REDUCEQN:

```
COMP-TOP = /TOP.

SUBINEQN(COMP-TOP, COMP-TOP).

REDUCEQN(COMP-TOP, COMP-TOP).
```

We can interpret the complement of an area as an area for which access is denied. Thus a minimal cut set of the complement equation can be interpreted as follows: if we deny access to all of the areas in the minimal cut set, then the top event of the sabotage fault tree cannot occur, i.e., /TOP will occur. Note that only one such minimal cut set of complements of areas is required, so we can choose the minimal cut set which represents the smallest number of areas to protect. Other criteria, such as the minimal cut set with a collection of vital areas which can be protected with the least impact on operability of the plant can also be applied. Of course, every Type 1 vital area will appear in every minimal cut set of the complement equation, so there are still certain vital areas that must be protected no matter what criterion is applied.

6.2 An Example of Using SETS to Find the Complement Equation

The following SETS program determines the complement of the vital area minimal cut set equation for the example in Section 5.4.

```
PROGRAM$EXAMPLE-5.
BLKSTAT.
LOBLK(VITAL-AREA-MCS).
VITAL-AREA-COMP = /VITAL-AREA-MCS.
SUBINEQN(VITAL-AREA-COMP,VITAL-AREA-COMP).
REDUCEQN(VITAL-AREA-COMP,VITAL-AREA-COMP).
FRMBLK(VITAL-AREA-COMP*ONLY$VITAL-AREA-COMP).
BLKSTAT.
PRTEQNONF(VITAL-AREA-COMP).
```

The BLKSTAT statement is used to verify that the block which contains the area minimal cut set equation is on the block file. The output of the BLKSTAT statement is:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  SABOTAGE-FT
                                                  2.  AREA-EQUATIONS
                                                  3.  OMEGA-ASSUMPTION
                                                  4.  VITAL-AREA-MCS
```

The LDBLK (VITAL-AREA-MCS) statement loads the vital area minimal cut set equation into the equation file. The Boolean equation VITAL-AREA-COMP = /VITAL-AREA-MCS establishes this additional equation in the equation file. The SUBINEQN statement substitutes for VITAL-AREA-COMP until the substitution process terminates on the variables which represent vital areas in the vital area minimal cut set equation. The REDUCEQN statement determines the minimal cut sets for VITAL-AREA-COMP and the FRMBLK statement forms a block of this complement equation. The PRTEQNDNF statement prints the minimal cut set equation for VITAL-AREA-COMP. The output is:

```
 TERM
NUMBER

        VITAL-AREA-COMP =

    1   /SFGRDB * /CR * /ESGRM * /2AB * /CHPMCUB * /RC +

    2   /SFGRDA * /MCC1JCV * /MCC1HCV * /CR * /ESGRM * /2AB * /CHPMCUB * /RC +

    3   /SFGRDA * /MCC1JCV * /MCC1HCV * /CR * /DGRM1 * /ESGRM * /WSGRM * /2AB *
        /TBSMTVVPT * /CHPMCUB * /RWST * /2RWST * /RLYRM * /3EQUIPRM * /PMPHS +

    4   /SFGRDB * /SFGRDA * /MCC1HCV * /CR * /DGRM1 * /ESGRM * /WSGRM * /2AB *
        /TBSMTVVPT * /CHPMCUB * /RWST * /2RWST * /RLYRM * /3EQUIPRM * /PMPHSE +

    5   /SFGRDA * /MCC1JCV * /MCC1HCV * /CR * /DGRM2 * /ESGRM * /WSGRM * /2AB *
        /TBSMTVVPT * /CHPMCUB * /RWST * /2RWST * /RLYRM * /3EQUIPRM * /PMPHS +

    6   /SFGRDB * /SFGRDA * /MCC1JCV * /CR * /DGRM2 * /ESGRM * /WSGRM * /2AB *
        /TBSMTVVPT * /CHPMCUB * /RWST * /2RWST * /RLYRM * /3EQUIPRM * /PMPHSE
```

6.3 A Potential Problem in Solving the Complement Equation

For some vital area minimal cut set equations, the complement equation is of a manageable size and can be determined as in the preceding example. However, sometimes the complement expression is so large that it cannot be simplified to minimal cut sets in a reasonable amount of computer time and/or computer storage. Since we are usually not interested in all of the minimal cut sets if there are thousands of them, but only in the minimal cut sets which represent feasible alternatives as sets of areas to protect, the approach to take is to determine only minimal cut sets up to some order. This can be done by using a truncation parameter, n, in the REDUCEQN statement which drops all cut sets with more than n variables (which represent areas in this case).

To determine the truncation value n for a particular problem, we first use REDUCEQN on the complement equation without a truncation parameter. A reasonable time limit on the computer run (about 50 seconds) should be used to prevent the REDUCEQN algorithm from executing for a great deal of time on an equation it cannot solve completely. When the time limit is reached, the output of the REDUCEQN procedure call will look something like this:

```
THE MAXIMUM NUMBER OF TERMS THAT CAN BE
GENERATED BY EXPANSION IS    14485500.
THE WORK MEASURE FOR EXPANSION IS    36792594.

TERMS GENERATED BY EXPANSION
      8 TERMS CONTAIN 20 LITERALS.
    115 TERMS CONTAIN 21 LITERALS.
    651 TERMS CONTAIN 22 LITERALS.
   1991 TERMS CONTAIN 23 LITERALS.
   3078 TERMS CONTAIN 24 LITERALS.
   2669 TERMS CONTAIN 25 LITERALS.
    668 TERMS CONTAIN 26 LITERALS.
      8 TERMS CONTAIN 27 LITERALS.
    153 TERMS CONTAIN 28 LITERALS.
  15321 TERMS CONTAIN 29 LITERALS.
  81703 TERMS CONTAIN 30 LITERALS.
 211711 TERMS CONTAIN 31 LITERALS.
```

The terms generated by expansion correspond to cut sets that must still be simplified to minimal cut sets. If we truncate this expression at n = 23, there will be 8 + 115 + 651 + 1991 = 2,765 cut sets which are then simplified to minimal cut sets. If the number of resulting minimal cut sets is sufficient to provide a reasonable number of alternative sets of vital areas to protect, we are finished. If not, we can truncate at a higher n, say n = 24, until a reasonable number of minimal cut sets is produced.

7.0 MINIMAL CUT SETS OF SABOTAGE ACTS

A minimal cut set of the sabotage fault tree in terms of vital areas is a combination of vital areas for which at least one collection of sabotage acts, each of which can be accomplished in at least one of the vital areas in the minimal cut set, will cause the top event of the fault tree to occur. Up until now, we have been concerned with areas only and have not identified any collections of sabotage acts which can cause the top event to occur. Such a collection of sabotage acts is called a scenario. The final analysis of a vital area analysis attempts to find all of the scenarios for the Type 1 vital areas.

The scenarios are minimal cut sets of the sabotage fault tree in terms of the fault tree primary events, i.e., sabotage acts. Thus, if we would solve the fault tree by the same SETS user program as described in Chapter 5, but without loading the area equations, we would obtain all possible scenarios which are in a minimal form, meaning that every sabotage act in the scenario is necessary for the top event to occur. However, the number of scenarios is often very large so that all of the scenarios cannot be obtained. Furthermore, there would be no indication of which vital areas are involved in each scenario. Fortunately, we are interested in only scenarios for the Type 1 vital areas which makes the problem much more manageable. Also, by employing a slightly different set of equations then the area equations, we can retain the information as to which vital area is involved in a scenario.

7.1 The SETS User Program for Identifying Scenarios

The SETS user program which determines the scenarios for a Type 1 vital area is similar to the program described in Chapter 5 for determining the vital areas. The differences are:

- Instead of loading the area equations, a different set of equations is loaded into the equation file.
- All of the areas which are not Type 1 vital areas are set to /OMEGA.
- The REDUCEQN(X, X) procedure calls are changed to REDUCEQN(X/1, X*EXCEPTCMP$).

In Chapter 4.0, the example given for an area equation was:

```
B1 = AREA1 + AREA2 + AREA3 * AREA4 + AREA5.
```

The corresponding equation for the identification of scenarios is:

```
B1 = (AREA1 + AREA2 + AREA3 * AREA4 + AREA5)*/B1Z.
```

This equation not only identifies the areas in which sabotage act B1 can occur, but it also maintains an explicit representation of

B1. The representation for B1 is /B1Z instead of B1. SETS does not allow the same variable to appear on both sides of an equation since a loop will occur during a SUBINEQN procedure call, so the letter Z, or any other symbol, is appended to B1. The '/' symbol is included in the new representation for B1 to take advantage of the EXCEPTCMP option of the REDUCEQN procedure. The REDUCEQN(B1/1, B1 * EXCEPTCMP$) statement will truncate any minimal cut set which involves more than one area (we are only interested in scenarios which can be completed in a single TYPE 1 vital area) but will allow any number of sabotage acts to be represented since it does not count any complemented events and all of the sabotage events are complemented.

The output of this SETS user program consists of minimal cut sets each of which are comprised of one Type 1 vital area and a collection of sabotage acts which represent one scenario for the vital area.

7.2 An Example of a SETS Program for Identifying the Scenarios of Type 1 Vital Areas

The following SETS user program determines the scenarios for the Type 1 vital areas of the example sabotage fault tree SABOTAGE-FT. The program is very similar to the program which solved SABOTAGE-FT for the vital area minimal cut sets in Section 5.4. Each step of the program is discussed in detail following the program listing.

```
PROGRAM$EXAMPLE-6.
RDBLK(SCENARIO-EQNS).
RDBLK(TYPE2-PHI).
BLKSTAT.
LDBLK(SABOTAGE-FT,SCENARIO-EQNS,TYPE2-PHI,OMEGA-ASSUMPTION).
SUBINEQN(AC-4160-B1J,AC-4160-B1J).
REDUCEQN(AC-4160-B1J/1,AC-4160-B1J*EXCEPTCMP$).
SUBINEQN(AC-4160-B1H,AC-4160-B1H).
REDUCEQN(AC-4160-B1H/1,AC-4160-B1H*EXCEPTCMP$).
SUBINEQN(SWA-IHR,SWA-IHR).
REDUCEQN(SWA-IHR/1,SWA-IHR*EXCEPTCMP$).
SUBINEQN(SWB-IHR,SWB-IHR).
REDUCEQN(SWB-IHR/1,SWB-IHR*EXCEPTCMP$).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL/1,CHO-P1A-COOL).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL/1,CHO-P1B-COOL).
SUBINEQN(ECRS-D,ECRS-D).
REDUCEQN(ECRS-D/1,ECRS-D*EXCEPTCMP$).
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L/1,AFO-PM-L*EXCEPTCMP$).
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L/1,CHO-PM-L*EXCEPTCMP$).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI/1,FM-TI*EXCEPTCMP$).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD/1,LI-MSD*EXCEPTCMP$).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP/1,SCENARIOS-TYPE1*EXCEPTCMP$).
FRMBLK(SCENARIOS-TYPE1*ONLY$SCENARIOS-TYPE1).
BLKSTAT.
PRTEQNDNF(SCENARIOS-TYPE1).
```

7.2.1 The Scenario Equations

The first statement in the program, RDBLK(SCENARIO-EQNS), reads the scenario equations. These equations are the same as the area equations in Section 4.3.2 except that an explicit representation of the sabotage act on the left-hand-side of the equation is included on the right-hand-side of the equation. The input block SCENARIO-EQNS is:

```
BLOCK$SCENARIO-EQNS.
IHR-NHRS                =(SFGRDB)*/IHR-NHRSZ.
RPS-D                   =(SFGRDA)*/RPS-DZ.
OI-RC                   =(SFGRDB)*/OI-RCZ.
SSRS-D                  =(MCC1JCV + MCC1HCV)*/SSRS-DZ.
AFO-PMD-HS  =(CR + MCC1JCV)*/AFO-PMD-HSZ.
AFO-CS-PMS  =(SFGRDA)*/AFO-CS-PMSZ.
AFO-P1A-B   =(MCC1HCV)*/AFO-P1A-BZ.
AFO-P1A-CSG  =(MCC1HCV)*/AFO-P1A-CSGZ.
AFO-P1A-COOL  =(DGRM2)*/AFO-P1A-COOLZ.
AFO-P1B-B   =(MCC1JCV)*/AFO-P1B-BZ.
AFO-P1B-CSG  =(MCC1JCV)*/AFO-P1B-CSGZ.
AFO-P1B-COOL  =(DGRM1 + ESGRM)*/AFO-P1B-COOLZ.
EP-BS-1J-D              =(ESGRM)*/EP-BS-1J-DZ.
CB-C1J-D                =(ESGRM)*/CB-C1J-DZ.
EP-BS-1H-D              =(ESGRM)*/EP-BS-1H-DZ.
CB-C1H-D                =(ESGRM)*/CB-C1H-DZ.
DG-NO3-B                =(DGRM2)*/DG-NO3-BZ.
DG-NO1-B                =(DGRM1)*/DG-NO1-BZ.
ID-BS-480H-D            =(WSGRM)*/ID-BS-480H-DZ.
ID-TR-SSXFM1H-D         =(2AB)*/ID-TR-SSXFM1H-DZ.
ID-BS-480J-D            =(ESGRM)*/ID-BS-480J-DZ.
ID-TR-SSXFM1J-D         =(3EQUIPRM)*/ID-TR-SSXFM1J-DZ.
AFWS-LO-IHR     =(CR+MCC1HCV*MCC1JCV+SFGRDA)*/AFWS-LO-IHRZ.
CS1A-PP-PMD-B           =(3EQUIPRM)*/CS1A-PP-PMD-BZ.
CS1A-CX-SUMP-B          =(CR)*/CS1A-CX-SUMP-BZ.
CS1A-PP-PMS-B           =(CR + 2AB)*/CS1A-PP-PMS-BZ.
CS1A-P1A-B              =(PMPHSE)*/CS1A-P1A-BZ.
CS1A-P1A-CSG            =(CR)*/CS1A-P1A-CSGZ.
CS1B-PP-PMD-B           =(CR)*/CS1B-PP-PMD-BZ.
CS1B-CX-SUMP-B          =(PMPHSE)*/CS1B-CX-SUMP-BZ.
CS1B-PP-PMS-B           =(CR + 2AB*2RB)*/CS1B-PP-PMS-BZ.
CS1B-P1B-B              =(CR)*/CS1B-P1B-BZ.
CS1B-P1B-CSG            =(CR)*/CS1B-P1B-CSGZ.
SWA-HX-OILCO-B          =(TBSMTVVPT)*/SWA-HX-OILCO-BZ.
SWA-PP-PMD-B            =(3EQUIPRM+ESGRM*WSGRM+2AB)*/SWA-PP-PMD-BZ.
SWA-VV-PMD              =(3EQUIPRM+2AB)*/SWA-VV-PMDZ.
SWB-HX-OILCO-B          =(CHPMCUB)*/SWB-HX-OILCO-BZ.
SWB-PP-PMD-B            =(3EQUIPRM+ESGRM*WSGRM+2AB)*/SWB-PP-PMD-BZ.
SWB-VV-PMD              =(3EQUIPRM+2AB)*/SWB-VV-PMDZ.
CHO-HX-RVESS-B          =(CR)*/CHO-HX-RVESS-BZ.
CHO-PP-PMD-B            =(CHPMCUB+2AB)*/CHO-PP-PMD-BZ.
```

```
CHO-MV1286ABCOC    =(CR+MCC1HCV*MCC1JCV+CHPMCUB)*/CHO-MV1286ABCOCZ.
CHO-MV1867ABCC     =(CR+MCC1HCV*MCC1JCV+2AB)*/CHO-MV1867ABCCZ.
CHO-MV1867BDCC     =(CR+MCC1JCV*MCC1HCV+2AB)*/CHO-MV1867BDCCZ.
CHO-P1A-B          =(CHPMCUB)*/CHO-P1A-BZ.
CHO-P1A-CSG        =(CHPMCUB)*/CHO-P1A-CSGZ.
CHO-P1B-B          =(ESGRM)*/CHO-P1B-BZ.
CHO-P1B-CSG        =(CHPMCUB)*/CHO-P1B-CSGZ.
LI   =( CHPMCUB + 2AB + ESGRM + CR + RC)*/LIZ.
CS1A-HS =( SFGRDA + MCC1HCV)*/CS1A-HSZ.
CS1B-HS =( SFGRDA + MCC1JCV + WSGRM)*/CS1B-HSZ.
SWA-CS-PMS =( TBSMTVVPT)*/SWA-CS-PMSZ.
SWB-CS-PMS =( TBSMTVVPT)*/SWB-CS-PMSZ.
SWA-MD-P10A =( CR + 3EQUIPRM)*/SWA-MD-P10AZ.
SWB-MD-P10B =( CR + 3EQUIPRM)*/SWB-MD-P10BZ.
CHO-CS-PMS =( RWST + 2RWST+CR + 2AB + CHPMCUB)*/CHO-CS-PMSZ.
DG-NO3-AUX =( DGRM2)*/DG-NO3-AUXZ.
DG-NO1-AUX =( DGRM1)*/DG-NO1-AUXZ.
DG-NO3-CSG =( CR + RLYRM + DGRM1*DGRM2)*/DG-NO3-CSGZ.
DG-NO1-CSG =( CR + RLYRM + DGRM1*DGRM2)*/DG-NO1-CSGZ.
```

7.2.2 Setting the Type 2 Vital Areas to /OMEGA

The second statement in the SETS user program, RDBLK (TYPE2-PHI), reads the set of equations which sets all of the Type 2 areas to /OMEGA. In Boolean algebra, /OMEGA = PHI, however, SETS will only recognize the /OMEGA representation in a Boolean equation. The input block TYPE2-PHI for the example is given by:

```
BLOCK$TYPE2-PHI.
SFGRDB = /OMEGA.
SFGRDA = /OMEGA.
MCC1JCV = /OMEGA.
MCC1HCV = /OMEGA.
DGRM2 = /OMEGA.
DGRM1 = /OMEGA.
WSGRM = /OMEGA.
3EQUIPRM = /OMEGA.
PMPHSE = /OMEGA.
2RB = /OMEGA.
TBSMTVVPT = /OMEGA.
RC = /OMEGA.
RWST = /OMEGA.
2RWST = /OMEGA.
RLYRM = /OMEGA.
```

The following BLKSTAT statement is used to verify the contents of the block file at this point in the analysis. For the example, the output of the BLKSTAT statement is:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  SABOTAGE-FT
                                                  2.  AREA-EQUATIONS
                                                  3.  OMEGA-ASSUMPTION
                                                  4.  VITAL-AREA-MCS
                                                  5.  VITAL-AREA-COMP
                                                  6.  SCENARIO-EQNS
                                                  7.  TYPE2-PHI
```

7.2.3 The Equation File

The LDBLK (SABOTAGE-FT, SCENARIO-EQNS, TYPE2-PHI, OMEGA-ASSUMPTION) statement loads the equations contained in these blocks into the equation file, in the order in which the blocks are listed in the LDBLK statement. The following equations are now in the equation file:

- The equations which represent the sabotage fault tree, as listed in Section 4.3.1.
- The scenario equations from Section 7.2.1.
- The equations which set the Type 2 vital areas to /OMEGA, given in Section 7.2.2.
- The events which are assumed to always occur or never occur, set to OMEGA or /OMEGA, respectively, as given in Section 4.3.3.

Note that the difference between this LDBLK and the LDBLK statement in the program which determined the vital area minimal cut sets is that the blocks which contain the scenario equations and the equations which set the Type 2 vital areas to /OMEGA are loaded instead of the area equations.

7.2.4 Solving the Sabotage Fault Tree for Scenarios

We now proceed with the SUBINEQN and REDUCEQN statements which solve the same gates, in the same order, as the SETS program in Section 5.4 which solved the fault tree for vital area minimal cut sets. The SUBINEQN statements are identical in the two programs. The REDUCEQN statements are modified to include the truncation parameter of 1 and the EXCEPTCMP option. The truncation parameter of 1 is used so we do not get scenarios for combinations of Type 1 vital areas. The EXCEPTCMP option is used since we do not want the sabotage acts to be counted when truncating; there may be any number of sabotage acts in a particular scenario. The last REDUCEQN statement is the one which reduces the cut sets of the top gate to minimal cut sets. The final name of the solution is the second parameter of this REDUCEQN, so it is called SCENARIOS-TYPE1 rather than VITAL-AREA-MCS as in the earlier program.

To summarize, the SUBINEQN and REDUCEQN statements are exactly the same, and in the same order, as the ones in the program used to solve the sabotage fault tree for vital areas except:

- A truncation parameter of 1 is added to each REDUCEQN.
- The EXCEPTCMP option is added to each REDUCEQN.
- The second parameter of the final REDUCEQN is given a suitable name for this solution.

Thus we are really solving the same fault tree, in the same order, over again. The equations in the equation file have been changed, however, so that the substitution process is terminating on combinations of Type 1 vital areas and sabotage acts, instead of on just vital areas as in the previous program in Section 5.4.

7.2.5 The Solution Equation

The FRMBLK (SCENARIOS-TYPE1 *ONLY$ SCENARIOS-TYPE1) statement forms a block containing only the final equation which identifies the scenarios for the Type 1 vital areas. The BLKSTAT is used to check the status of the block file which is:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS
                                        1.  SABOTAGE-FT
                                        2.  AREA-EQUATIONS
                                        3.  OMEGA-ASSUMPTION
                                        4.  VITAL-AREA-MCS
                                        5.  VITAL-AREA-COMP
                                        6.  SCENARIO-EQNS
                                        7.  TYPE2-PHI
                                        8.  SCENARIOS-TYPE1
```

Finally, the solution equation is printed by the PRTEQNDNF statement. The output is:

* * * * LITERAL OCCURRENCE TABLE * * * *

LITERAL	NUMBER OF OCCURRENCES	OPPOSITION LITERAL	NUMBER OF OCCURRENCES
CR	25		
ESGRM	8		
		/EP-BS-1J-DZ	2
		/CB-C1J-OZ	2
		/EP-BS-1H-DZ	4
		/CB-C1H-OZ	4
2AB	4		
		/ID-BS-480J-DZ	2
		/CS1A-CX-SUMP-BZ	5
		/CS1A-PP-PMS-BZ	5
		/CS1A-P1A-CSGZ	5
		/CS1B-PP-PMD-BZ	4
		/CS1B-PP-PMS-BZ	4
		/CS1B-P1B-BZ	4
		/CS1B-P1B-CSGZ	4
CHPMCUB	5		
		/CHO-HX-[illegible]CSS-BZ	1
		/CHO-PP-PMD-BZ	2
		/CHO-MV1286ABCOCZ	2
		/CHO-MV1867ABCCZ	2
		/CHO-MV1867BDCCZ	2
		/CHO-P1A-BZ	1
		/CHO-P1A-CSGZ	1
		/CHO-P1B-BZ	2
		/CHO-P1B-CSGZ	2
		/LIZ	42
		/CHO-CS-PMSZ	3
		/DG-NO3-CSGZ	4
		/DG-NO1-CSGZ	5

THERE ARE 29 DIFFERENT LITERALS IN THE EQUATION FOR SCENARIOS-TYPE1

TERM NUMBER	NUMBER OF LITERALS	
		SCENARIOS-TYPE1 =
1	3	CR * /LIZ * /CHO-CS-PMSZ +
2	3	CR * /CHO-MV1867BDCCZ * /LIZ +
3	3	CR * /CHO-MV1867ABCCZ * /LIZ +
4	3	CR * /CHO-MV1286AB[illegible]OCZ * /LIZ +

TERM NUMBER	NUMBER OF LITERALS	
5	3	CR * /CHO-HX-RVESS-BZ * /LIZ +
6	3	CHPMCUB * /LIZ * /CHO-CS-PMSZ +
7	3	CHPMCUB * /CHO-MV1286ABCOCZ * /LIZ +
8	3	CHPMCUB * /CHO-PP-PMD-BZ * /LIZ +
9	3	2AB * /LIZ * /CHO-CS-PMSZ +
10	3	2AB * /CHO-MV1867BDCCZ * /LIZ +
11	3	2AB * /CHO-MV1867ABCCZ * /LIZ +
12	3	2AB * /CHO-PP-PMD-BZ * /LIZ +
13	4	CR * /LIZ * /DG-NO3-CSGZ * /DG-NO1-CSGZ +
14	4	CR * /CS1B-P1B-CSGZ * /LIZ * /DG-NO1-CSGZ +
15	4	CR * /CS1B-P1B-BZ * /LIZ * /DG-NO1-CSGZ +
16	4	CR * /CS1B-PP-PMS-BZ * /LIZ * /DG-NO1-CSGZ +
17	4	CR * /CS1B-PP-PMD-BZ * /LIZ * /DG-NO1-CSGZ +
18	4	CR * /CS1A-P1A-CSGZ * /LIZ * /DG-NO3-CSGZ +
19	4	CR * /CS1A-P1A-CSGZ * /CS1B-P1B-CSGZ * /LIZ +
20	4	CR * /CS1A-P1A-CSGZ * /CS1B-P1B-BZ * /LIZ +
21	4	CR * /CS1A-P1A-CSGZ * /CS1B-PP-PMS-BZ * /LIZ +
22	4	CR * /CS1A-P1A-CSGZ * /CS1B-PP-PMD-BZ * /LIZ +
23	4	CR * /CS1A-PP-PMS-BZ * /LIZ * /DG-NO3-CSGZ +
24	4	CR * /CS1A-PP-PMS-BZ * /CS1B-P1B-CSGZ * /LIZ +
25	4	CR * /CS1A-PP-PMS-BZ * /CS1B-P1B-BZ * /LIZ +

TERM NUMBER	NUMBER OF LITERALS	
26	4	CR * /CS1A-PP-PMS-BZ * /CS1B-PP-PMS-BZ * /LIZ +
27	4	CR * /CS1A-PP-PMS-BZ * /CS1B-PP-PMD-BZ * /LIZ +
28	4	CR * /CS1A-CX-SUMP-BZ * /LIZ * /DG-NO3-CSGZ +
29	4	CR * /CS1A-CX-SUMP-BZ * /CS1B-P1B-CSGZ * /LIZ +
30	4	CR * /CS1A-CX-SUMP-BZ * /CS1B-P1B-BZ * /LIZ +
31	4	CR * /CS1A-CX-SUMP-BZ * /CS1B-PP-PMS-BZ * /LIZ +
32	4	CR * /CS1A-CX-SUMP-BZ * /CS1B-PP-PMD-BZ * /LIZ +
33	4	ESGRM * /CB-C1H-OZ * /CHO-P1B-BZ * /LIZ +
34	4	ESGRM * /CB-C1H-OZ * /ID-BS-480J-DZ * /LIZ +
35	4	ESGRM * /CB-C1J-OZ * /CB-C1H-OZ * /LIZ +
36	4	ESGRM * /EP-BS-1J-DZ * /CB-C1H-OZ * /LIZ +
37	4	ESGRM * /EP-BS-1H-DZ * /CHO-P1B-BZ * /LIZ +
38	4	ESGRM * /EP-BS-1H-DZ * /ID-BS-480J-DZ * /LIZ +
39	4	ESGRM * /CB-C1J-OZ * /EP-BS-1H-DZ * /LIZ +
40	4	ESGRM * /EP-BS-1J-DZ * /EP-BS-1H-DZ * /LIZ +
41	4	CHPMCUB * /CHO-P1A-CSGZ * /CHO-P1B-CSGZ * /LIZ +
42	4	CHPMCUB * /CHO-P1A-BZ * /CHO-P1B-CSGZ * /LIZ

The first minimal cut set, 2AB*/LIZ*/CHO-CS-PMSZ tells us that one successful scenario which can be accomplished in Type 1 vital area 2AB is to commit sabotage acts LI and CHO-CS-PMS (in their original notation). The remaining 48 minimal cut sets are interpreted similarly.

7.3 Potential Problems in Determining the Scenarios

If the fault tree has any troublesome replicated subtrees or AND gates, these will already have been dealt with in the program which solved the fault tree for vital area minimal cut sets. All of the additional SUBINEQN and REDUCEQN procedure calls to deal with these problems should be left in for the program that determines scenarios, with the REDUCEQN statements modified by adding the truncation value of 1 and the EXCEPTCMP option. In other words, this program should already deal with all of the problem areas of the fault tree by virtue of being essentially the same program as the one which solved the fault tree for vital area minimal cut sets. So the only problem likely to be encountered is that the number of all scenarios for the Type 1 vital areas is so large that we would rather get the scenarios for just one vital area at a time, particularly if the vital area is something like the control room which typically has a large number of scenarios.

When identifying the Type 1 vital areas we would like to process individually, it is helpful to have a list of all of the sabotage acts which can be accomplished in each Type 1 vital area. This information may have other uses for the analyst as well. Recall that the area equations tell us areas in which a sabotage act can be accomplished. We now wish to know which sabotage acts can be accomplished in an area. If the area equations are thought of as a mapping (or transformation) from sabotage acts to areas, we now seek the reverse mapping (or transformation) from areas to sabotage acts. For the example problem, the reverse mapping from areas to sabotage acts is given by:

LOCATIONS FOR VITAL AREA EQUATIONS

NO.OF LOCATIONS = 19

NO.OF EVENTS	LOCATIONS	EVENTS
2	SFGRDB	
		IHR-NHRS
		OI-RC
5	SFGRDA	
		RPS-D
		AFO-CS-PMS
		AFWS-LO-IHR
		CS1A-HS
		CS1B-HS
9	MCC1JCV	
		SSRS-D
		AFO-PMD-HS
		AFO-P1B-B
		AFO-P1B-CSG
		AFWS-LO-IHR
		CHO-MV1286ABCOC
		CHO-MV1867ABCC
		CHO-MV1867BDCC
		CS1B-HS
8	MCC1HCV	
		SSRS-D
		AFO-P1A-B
		AFO-P1A-CSG
		AFWS-LO-IHR
		CHO-MV1286ABCOC
		CHO-MV1867ABCC
		CHO-MV1867BDCC
		CS1A-HS
19	CR	
		AFO-PMD-HS
		AFWS-LO-IHR
		CS1A-CX-SUMP-B
		CS1A-PP-PMS-B
		CS1A-P1A-CSG
		CS1B-PP-PMD-B
		CS1B-PP-PMS-B
		CS1B-P1B-B
		CS1B-P1B-CSG
		CHO-HX-RVESS-B
		CHO-MV1286ABCOC
		CHO-MV1867ABCC
		CHO-MV1867BDCC
		LI
		SWA-MD-P10A
		SWB-MD-P10B
		CHO-CS-PMS
		DG-NO3-CSG
		DG-NO1-CSG
5	DGRM2	
		AFO-P1A-COOL
		DG-NO3-B
		DG-NO3-AUX
		DG-NO3-CSG

LOCATIONS FOR VITAL AREA EQUATIONS

NO.OF LOCATIONS = 19

NO.OF EVENTS	LOCATIONS	EVENTS
		DG-NO1-CSG
5	DGRM1	
		AFO-P1B-COOL
		DG-NO1-B
		DG-NO1-AUX
		DG-NO3-CSG
		DG-NO1-CSG
10	ESGRM	
		AFO-P1B-COOL
		EP-BS-1J-D
		CB-C1J-O
		EP-BS-1H-D
		CB-C1H-O
		ID-BS-480J-D
		SWA-PP-PMD-B
		SWB-PP-PMD-B
		CHO-P1B-B
		LI
4	WSGRM	
		ID-BS-480H-D
		SWA-PP-PMD-B
		SWB-PP-PMD-B
		CS1B-HS
12	2AB	
		ID-TR-SSXFM1H-D
		CS1A-PP-PMS-B
		CS1B-PP-PMS-B
		SWA-PP-PMD-B
		SWA-VV-PMD
		SWB-PP-PMD-B
		SWB-VV-PMD
		CHO-PP-PMD-B
		CHO-MV1867ABCC
		CHO-MV1867BDCC
		LI
		CHO-CS-PMS
8	3EQUIPRM	
		ID-TR-SSXFM1J-D
		CS1A-PP-PMD-B
		SWA-PP-PMD-B
		SWA-VV-PMD
		SWB-PP-PMD-B
		SWB-VV-PMD
		SWA-MD-P10A
		SWB-MD-P10B
2	PMPHSE	
		CS1A-P1A-B
		CS1B-CX-SUMP-B
1	2RB	
		CS1B-PP-PMS-B
3	TBSMTVVPT	
		SWA-HX-OILCU-B
		SWA-CS-PMS

LOCATIONS FOR VITAL AREA EQUATIONS

NO.OF LOCATIONS = 19

NO.OF EVENTS	LOCATIONS	EVENTS
		SWB-CS-PMS
8	CHPMCUB	
		SWB-HX-OILCO-B
		CHO-PP-PMO-B
		CHO-MV1286ABCOC
		CHO-P1A-B
		CHO-P1A-CSG
		CHO-P1B-CSG
		LI
		CHO-CS-PMS
1	RC	
		LI
1	RWST	
		CHO-CS-PMS
1	2RWST	
		CHO-CS-PMS
2	RLYRM	
		DG-NO3-CSG
		DG-NO1-CSG

7.3.1 Determining the Scenarios of a Single Type 1 Vital Area

If a Type 1 vital area has a relatively large number of sabotage acts like the control room CR here in our example, we may want to determine its scenarios without including the other Type 1 vital area scenarios. To accomplish this, a slight modification to the SETS program that finds all scenarios for all the Type 1 vital areas is required. After the LDBLK procedure call which loads all of the equations into the equation file, we establish equations which set all of the Type 1 vital areas to /OMEGA except the one we wish to find the scenarios for, in this case CR. The remaining Type 1 vital areas; 2AB, CHPMCUB and ESGRM are set to /OMEGA by Boolean equations. The SETS statements:

```
2AB = /OMEGA.
CHPMCUB = /OMEGA.
ESGRM = /OMEGA.
```

establish these equations in the equation file. The only other change in the program is to change the name of the solution from SCENARIOS-TYPE1 to SCENARIOS-CR in the final REDUCEQN statement and in the FRMBLK and PRTEQNDNF statements. Thus the SETS program to find the scenarios for just CR is as follows:

```
PROGRAM$EXAMPLE-7.
BLKSTAT.
LDBLK(SABOTAGE-FT,SCENARIO-EQNS,TYPE2-PHI,OMEGA-ASSUMPTION).
ESGRM = /OMEGA.
2AB   = /OMEGA.
CHPMCUB = /OMEGA.
SUBINEQN(AC-4160-B1J,AC-4160-B1J).
REDUCEQN(AC-4160-B1J/1,AC-4160-B1J*EXCEPTCMP$).
SUBINEQN(AC-4160-B1H,AC-4160-B1H).
REDUCEQN(AC-4160-B1H/1,AC-4160-B1H*EXCEPTCMP$).
SUBINEQN(SWA-IHR,SWA-IHR).
REDUCEQN(SWA-IHR/1,SWA-IHR*EXCEPTCMP$).
SUBINEQN(SWB-IHR,SWB-IHR).
REDUCEQN(SWB-IHR/1,SWB-IHR*EXCEPTCMP$).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL/1,CHO-P1A-COOL).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL/1,CHO-P1B-COOL).
SUBINEQN(ECRS-D,ECRS-D).
REDUCEQN(ECRS-D/1,ECRS-D*EXCEPTCMP$).
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L/1,AFO-PM-L*EXCEPTCMP$).
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L/1,CHO-PM-L*EXCEPTCMP$).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI/1,FM-TI*EXCEPTCMP$).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD/1,LI-MSD*EXCEPTCMP$).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP/1,SCENARIOS-CR*EXCEPTCMP$).
FRMBLK(SCENARIOS-CR*ONLY$SCENARIOS-CR).
BLKSTAT.
PRTEQNDNF(SCENARIOS-CR).
```

The output of the PRTEQNDNF statement identifies the scenarios for CR:

```
                SCENARIOS-CR =
1       3       CR * /LIZ * /CHO-CS-PMSZ +
2       3       CR * /CHO-MV1867BDCCZ * /LIZ +
3       3       CR * /CHO-MV1867ABCCZ * /LIZ +
4       3       CR * /CHO-MV1286ABCOCZ * /LIZ +
5       3       CR * /CHO-HX-RVESS-BZ * /LIZ +
6       4       CR * /LIZ * /DG-NO3-CSGZ * /DG-NO1-CSGZ +
7       4       CR * /CS1B-P1B-CSGZ * /LIZ * /DG-NO1-CSGZ +
8       4       CR * /CS1B-P1B-BZ * /LIZ * /DG-NO1-CSGZ +
9       4       CR * /CS1B-PP-PMS-BZ * /LIZ * /DG-NO1-CSGZ +
10      4       CR * /CS1B-PP-PMD-BZ * /LIZ * /DG-NO1-CSGZ +
11      4       CR * /CS1A-P1A-CSGZ * /LIZ * /DG-NO3-CSGZ +
12      4       CR * /CS1A-P1A-CSGZ * /CS1B-P1B-CSGZ * /LIZ +
13      4       CR * /CS1A-P1A-CSGZ * /CS1B-P1B-BZ * /LIZ +
14      4       CR * /CS1A-P1A-CSGZ * /CS1B-PP-PMS-BZ * /LIZ +
15      4       CR * /CS1A-P1A-CSGZ * /CS1B-PP-PMD-BZ * /LIZ +
16      4       CR * /CS1A-PP-PMS-BZ * /LIZ * /DG-NO3-CSGZ +
17      4       CR * /CS1A-PP-PMS-BZ * /CS1B-P1B-CSGZ * /LIZ +
18      4       CR * /CS1A-PP-PMS-BZ * /CS1B-P1B-BZ * /LIZ +
19      4       CR * /CS1A-PP-PMS-BZ * /CS1B-PP-PMS-BZ * /LIZ +
20      4       CR * /CS1A-PP-PMS-BZ * /CS1B-PP-PMD-BZ * /LIZ +
21      4       CR * /CS1A-CX-SUMP-BZ * /LIZ * /DG-NO3-CSGZ +
22      4       CR * /CS1A-CX-SUMP-BZ * /CS1B-P1B-CSGZ * /LIZ +
23      4       CR * /CS1A-CX-SUMP-BZ * /CS1B-P1B-BZ * /LIZ +
24      4       CR * /CS1A-CX-SUMP-BZ * /CS1B-PP-PMS-BZ * /LIZ +
25      4       CR * /CS1A-CX-SUMP-BZ * /CS1B-PP-PMD-BZ * /LIZ
```

Once the scenarios for the control room have been determined, we can find the scenarios for the remaining Type 1 vital areas as a group or one at a time. If we want to solve the remaining Type 1 vital areas as a group, we replace the equations:

```
2AB = /OMEGA.
CHPMCUB = /OMEGA.
ESGRM = /OMEGA.
```

by the equation:

```
CR = /OMEGA.
```

We also change the name of the solution equation in the final REDUCEQN procedure call and the FRMBLK and PRTEQNDNF statements, and rerun the program. For the example, the SETS program to identify the scenarios for the Type 1 vital areas except CR is given by:

```
PROGRAM$EXAMPLE-8.
BLKSTAT.
LDBLK(SABOTAGE-FT,SCENARIO-EQNS,TYPE2-PHI,OMEGA-ASSUMPTION).
CR    = /OMEGA.
SUBINEQN(AC-4160-B1J,AC-4160-B1J).
REDUCEQN(AC-4160-B1J/1,AC-4160-B1J*EXCEPTCMP$).
SUBINEQN(AC-4160-B1H,AC-4160-B1H).
REDUCEQN(AC-4160-B1H/1,AC-4160-B1H*EXCEPTCMP$).
SUBINEQN(SWA-IHR,SWA-IHR).
REDUCEQN(SWA-IHR/1,SWA-IHR*EXCEPTCMP$).
SUBINEQN(SWB-IHR,SWB-IHR).
REDUCEQN(SWB-IHR/1,SWB-IHR*EXCEPTCMP$).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL/1,CHO-P1A-COOL).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL/1,CHO-P1B-COOL).
SUBINEQN(ECRS-D,ECRS-D).
REDUCEQN(ECRS-D/1,ECRS-D*EXCEPTCMP$).
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L/1,AFO-PM-L*EXCEPTCMP$).
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L/1,CHO-PM-L*EXCEPTCMP$).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI/1,FM-TI*EXCEPTCMP$).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD/1,LI-MSD*EXCEPTCMP$).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP/1,SCENARIOS-REST*EXCEPTCMP$).
FRMBLK(SCENARIOS-REST*ONLY$SCENARIOS-REST).
BLKSTAT.
PRTEQNDNF(SCENARIOS-REST).
```

The output of the PRTEQNDNF prints the scenarios for the Type 1 vital areas except CR:

TERM NUMBER	NUMBER OF LITERALS	
		SCENARIOS-REST =
1	3	CHPMCUB * /LIZ * /CHO-CS-PMSZ +
2	3	CHPMCUB * /CHO-MV1286ABCOCZ * /LIZ +
3	3	CHPMCUB * /CHO-PP-PMD-BZ * /LIZ +
4	3	2AB * /LIZ * /CHO-CS-PMSZ +
5	3	2AB * /CHO-MV1867BDCCZ * /LIZ +
6	3	2AB * /CHO-MV1867ABCCZ * /LIZ +
7	3	2AB * /CHO-PP-PMD-BZ * /LIZ +
8	4	ESGRM * /CB-C1H-OZ * /CHO-P1B-BZ * /LIZ +
9	4	ESGRM * /CB-C1H-OZ * /ID-BS-480J-OZ * /LIZ +
10	4	ESGRM * /CB-C1J-OZ * /CB-C1H-OZ * /LIZ +
11	4	ESGRM * /EP-BS-1J-OZ * /CB-C1H-OZ * /LIZ +
12	4	ESGRM * /EP-BS-1H-OZ * /CHO-P1B-BZ * /LIZ +
13	4	ESGRM * /EP-BS-1H-OZ * /ID-BS-480J-OZ * /LIZ +
14	4	ESGRM * /CB-C1J-OZ * /EP-BS-1H-OZ * /LIZ +
15	4	ESGRM * /EP-BS-1J-OZ * /EP-BS-1H-OZ * /LIZ +
16	4	CHPMCUB * /CHO-P1A-CSGZ * /CHO-P1B-CSGZ * /LIZ +
17	4	CHPMCUB * /CHO-P1A-BZ * /CHO-P1B-CSGZ * /LIZ

We may choose to solve all of the Type 1 vital areas individually. This is done in the same way that the scenarios for CR were determined. An individual run is made for each Type 1 vital area with all of the other Type 1 vital areas set to /OMEGA, as was done in the example for CR.

7.3.2 Determining a Subset of Scenarios

Occasionally, a single vital area such as the control room may have a large number of scenarios, say 10,000. To determine all of these scenarios is a rather costly computer run and it is unlikely that anyone is interested in examining all 10,000 scenarios. We can obtain a subset of the scenarios, at much less cost, by only determining scenarios with n or fewer sabotage acts (a typical value for n is 6). If we change the truncation value from 1 to n + 1 (n + 1 because the vital area is counted as one literal) and remove the EXCEPTCMP option from each REDUCEQN statement in the program, then we will obtain all scenarios for the vital area with no more than n sabotage acts since the truncation criteria no longer excludes the sabotage acts from the counting process. For example, the following SETS program identifies the scenarios for the control room, CR, with less than three sabotage acts.

```
PROGRAM$EXAMPLE-9.
BLKSTAT.
LDBLK(SABOTAGE-FT,SCENARIO-EQNS,TYPE2-PHI,OMEGA-ASSUMPTION).
ESGRM = /OMEGA.
2AB   = /OMEGA.
CHPMCUB = /OMEGA.
SUBINEQN(AC-4160-B1J,AC-4160-B1J).
REDUCEQN(AC-4160-B1J/3,AC-4160-B1J).
SUBINEQN(AC-4160-B1H,AC-4160-B1H).
REDUCEQN(AC-4160-B1H/3,AC-4160-B1H).
SUBINEQN(SWA-IHR,SWA-IHR).
REDUCEQN(SWA-IHR/3,SWA-IHR).
SUBINEQN(SWB-IHR,SWB-IHR).
REDUCEQN(SWB-IHR/3,SWB-IHR).
SUBINEQN(CHO-P1A-COOL,CHO-P1A-COOL).
REDUCEQN(CHO-P1A-COOL/3,CHO-P1A-COOL).
SUBINEQN(CHO-P1B-COOL,CHO-P1B-COOL).
REDUCEQN(CHO-P1B-COOL/3,CHO-P1B-COOL).
SUBINEQN(ECRS-D,ECRS-D).
REDUCEQN(ECRS-D/3,ECRS-D).
SUBINEQN(AFO-PM-L,AFO-PM-L).
REDUCEQN(AFO-PM-L/3,AFO-PM-L).
SUBINEQN(CHO-PM-L,CHO-PM-L).
REDUCEQN(CHO-PM-L/3,CHO-PM-L).
SUBINEQN(FM-TI,FM-TI).
REDUCEQN(FM-TI/3,FM-TI).
SUBINEQN(LI-MSD,LI-MSD).
REDUCEQN(LI-MSD/3,LI-MSD).
SUBINEQN(TOP,TOP).
REDUCEQN(TOP/3,SCENARIOS-CR-3).
FRMBLK(SCENARIOS-CR-3*ONLY$SCENARIOS-CR-3).
BLKSTAT.
PRTEQNDNF(SCENARIOS-CR-3).
```

The output of this program identifies all scenarios for the control room which require fewer than three sabotage acts:

TERM NUMBER	NUMBER OF LITERALS	
		SCENARIOS-CR-3 =
1	3	CR * /LIZ * /CHO-CS-PMSZ +
2	3	CR * /CHO-MV18678DCCZ * /LIZ +
3	3	CR * /CHO-MV1867ABCCZ * /LIZ +
4	3	CR * /CHO-MV1286ABCOCZ * /LIZ +
5	3	CR * /CHO-HX-RVESS-BZ * /LIZ

7.3.3 Using Truncation when the STOP Option is Being Used

If there are any gates in the sabotage fault tree which are being solved using stop points, as described in Section 5.5.2, the stop points should not be counted toward the truncation value since we want to count only the vital area toward the truncation value. To avoid counting the stop points toward the truncation value, the EXCEPTNONCMP (except non-complemented variables) is used. The names of the gates which are stop points follow the EXCEPTNONCMP statement. For example, the SETS user program to solve gate X in Section 5.5.2 for scenarios of Type 1 vital areas would be altered as follows:

```
SUBINEQN(X,X*STOP$C,D,E).
REDUCEQN(X/1,X*EXCEPTCMP$/EXCEPTNONCMP$C,D,E).
SUBINEQN(X,X*STOP$D,E).
REDUCEQN(X/1,X*EXCEPTCMP$/EXCEPTNONCMP$D,E).
SUBINEQN(X,X*STOP$E).
REDUCEQN(X/1,X*EXCEPTCMP$/EXCEPTNONCMP$E).
SUBINEQN(X,X).
REDUCEQN(X/1,X*EXCEPTCMP$).
```

APPENDIX A

Procedures Available in SETS

Each of the procedures available in SETS is invoked by a procedure call statement in a SETS user program. A procedure call begins with a procedure identifier and is usually followed by a parameter part that is enclosed in parentheses. There are options that can be specified in the calls for some of the procedures which affect the processing that is achieved with those procedures. Some of the options involve the concepts of phi and omega. In the context of set theory, phi represents the empty set (Φ), and omega represents the universal set (Ω); while in the context of Boolean algebra, phi = 0 and omega = 1. The processing that is accomplished by the execution of a procedure and any options that can be used to affect that processing will be described for each procedure discussed.

A.1 Read Block

A call of the Read Block procedure has the form:

RDBLK (b_1, b_2, ..., b_k).

This procedure is used to read blocks. The parameters b_1, b_2, ..., b_k are the names of the blocks that are to be read. The blocks must be supplied as input in the same left to right order that the block names occur as parameters in the procedure call.

A block is a set of Boolean equations that can be read by the Read Block procedure. A block is comprised of a block header and a group of one or more Boolean equations. The block header precedes the equations and has the form:

BLOCK$ block-name.

where

"block-name" is the name of the block.

Each equation in the block must be terminated with a period.

The equations in a block are checked as they are read to ensure that they are correctly formed equations. After each block has been read and the equations have been checked, the block is added to the block file.

A.2 Read Fault Tree

A call of the Read Fault Tree procedure has the form:

RDFT (ft_1, ft_2, ..., ft_k).

This procedure is used to read fault trees. The parameters ft_1, ft_2, ..., ft_k are the names of the fault trees that are to be read. The fault trees must be supplied as input in the same left to right order that the fault tree names occur as parameters in the procedure call.

The fault tree input that was defined in Chapter 2 is a data structure that can be read by the Read Fault Tree procedure. The redundancy inherent in the input representation of a fault tree is used to check the structure of each fault tree as it is read and processed. After each fault tree is read and checked, a block is created for that fault tree and added to the block file. The block contains the intermediate event equations for the fault tree, and the block name is the same as the fault tree name.

Each block that is generated by the Read Fault Tree procedure contains a representation of the fault tree in addition to the equations that are contained in the block. This internal representation of the fault tree is used to produce the Fault Tree Event Table when the fault tree is printed using the Print Block procedure.

A.3 Print Equation

A call of the Print Equation procedure has the form:

PRTEQN (v_1, v_2, ..., v_k).

This procedure is used to print equations that are in the equation file. The parameters v_1, v_2, ..., v_k are processed from left to right and the equation for each variable is printed as it is encountered. If the equation file does not contain an equation for a particular v_i, the message

THERE IS NO SET EQUATION FOR v_i

is printed.

The equations in the equation file are in a factored form, and they are printed in this form by the Print Equation procedure. If there are any parentheses in an equation, an integer will be printed immediately below each parenthesis when the equation is printed. The numbers are provided to aid in the interpretation of complex equations. Paired parentheses have the same number and the numbering begins with the number 1 for an outermost set of parentheses. In a printed equation, the operations of AND, OR, and NOT are represented by * , + , and / , respectively.

A.4 Print Equation In Disjunctive Normal Form

A call of the Print Equation in Disjunctive Normal Form procedure has the form:

PRTEQNDNF (p_1, p_2, ..., p_k).

This procedure is used to print equations that are in the equation file. Each of the parameters p_1, p_2, ..., p_k is either a variable name v_i, or it is a variable name and a truncation value of the form v_i/n, where n is a positive integer. The parameters are processed from left to right and the equation for each variable is printed as it is encountered. If the equation file does not contain an equation for a particular v_i, the message

THERE IS NO SET EQUATION FOR v_i

is printed.

When a truncation value is specified, only those terms of the equation with n or fewer variables are printed. If every term of an equation contains more than n variables, the message

THE SET EQUATION IS PHI

is printed.

A Literal Occurrence Table is printed preceding each equation that is printed. The table indicates the number of times that a variable (literal) occurs in the printed equation. Since the equation is printed in a disjunctive normal form, the number of occurrences of a variable is also the number of terms which contain the variable. If any terms of an equation are discarded because of a truncation value, some variables that occur in the full equation may not occur in the truncated equation that is printed. The Literal Occurrence Table contains a count of only those variables which occur in the printed equation.

Following the Literal Occurrence Table for an equation, the equation is printed in a disjunctive normal form. The terms are numbered and they are printed in the order of an increasing number of variables per term.

A.5 Delete Equation

A call of the Delete Equation procedure has one of the forms:

a. DLTEQN.

b. DLTEQN (v_1, v_2, ..., v_k).

This procedure is used to delete equations from the equation file. If there is no parameter list in the procedure call (form a.), every equation is deleted from the equation file. When a parameter list occurs in the call (form b.), only the equations for the variables v_1, v_2, ..., v_k are deleted from the equation file. If there is no equation in the equation file for a particular variable v_i, then no action is taken for that parameter.

A.6 Substitute In Equation

A call of the Substitute In Equation procedure has one of the forms:

a. SUBINEQN (v_1, v_2).

b. SUBINEQN (v_1, v_2 * $o_1/o_2/o_3$).

This procedure is used to create a new equation and enter it into the equation file. The right side of the new equation is generated from the equation for v_1 by a repeated process of substituting equals for equals. The left side variable for the new equation is v_2.

For both forms of the procedure call, a copy of the equation for the first parameter, v_1, is used to start the substitution process. If there is no equation for v_1 in the equation file, then v_1 is taken as the right side expression for the new equation. If there is an equation for v_1 in the equation file, then each variable in the right side expression of the equation for v_1 which has an equation in the equation file, is replaced by the right side of the equation for that variable. By repeating this substitution process for every variable in the right side expression, including variables that have been introduced by a prior substitution, the expression will ultimately contain only variables for which there is no equation in the equation file and no further substitutions can be made.

If there are no substitution control options in the procedure call (form a.), the substitution process will terminate when none of the variables remaining in the expression have an equation in the equation file. However, if substitution control options occur in the call (form b.), these options are used to arrest the substitution process prior to its normal completion. The parameters o_1, o_2, and o_3 represent the three options that can occur in this form of the procedure call.

An omega option has the form:

OMEGA\$ v_1, v_2, ..., v_k

The omega option causes every occurrence of each v_i to be replaced by the variable OMEGA rather than the right side of the equation for v_i. The equation for v_i in the equation file is not affected.

A phi option has the form:

PHI\$ v_1, v_2, ..., v_k

The phi option causes every occurrence of each v_i to be replaced by the variable $\neg$ OMEGA rather than the right side of the equation for v_i. The equation for v_i in the equation file is not affected.

A stop option has the form:

STOP$ v_1, v_2, ..., v_k

The stop option causes every occurrence of each v_i to be treated as if there is no equation for v_i in the equation file (ie, no substitution for v_i will take place), and v_i will remain in the expression. The equation for v_i in the equation file is not affected. One or more of the options o_1, o_2, or o_3 can occur in the procedure call separated by "/" delimiters. Moreover, the options can occur in any order.

A.7 Reduce Equation

A call of the Reduce Equation procedure has one of the forms:

a. REDUCEQN (v_1, v_2).

b. REDUCEQN (v_1, v_2 * o_1/o_2).

c. REDUCEQN (v_1/n, v_2).

d. REDUCEQN (v_1/n, v_2 * $o_1/o_2/o_3/o_4$).

This procedure is used to create a new equation and enter it into the equation file. The right side of the new equation is generated by applying certain Boolean identities to the right side of the equation for v_1. The left side variable for the new equation is v_2.

The processing by the Reduce Equation procedure is concerned primarily with the reduction of a Boolean expression. During the processing, the form of the expression changes from a factored form, to a disjunctive normal form, and then back again to a factored form. The processing begins with a copy of the right side expression from the equation for v_1 and is achieved in three steps:

1. Expansion

   a. Apply DeMorgan's Rules to the factored form of the expression to eliminate NOT operators.

   b. Repeatedly apply the distributive law to the factored form of the expression to generate a disjunctive normal form of the expression.

   c. Apply the identities $P*P = P$ and $P*\neg P = \Phi$ to the expression to eliminate repeated variables in a term and terms with zero products.

2. Simplification

   Apply the identity P + P*Q = P to the disjunctive normal form of the expression to eliminate terms that are logically contained in other terms (absorption rule).

3. Factorization

   Factor (group) the disjunctive normal form of the expression to create a factored form of the reduced expression. (The factoring scheme is based on choosing as a factor the most often occurring variable whenever a factor is selected.)

For all forms of the procedure call, a copy of the right side expression from the equation for v_1 is expanded, simplified, and factored to form the right side of the new equation. If there is no equation for v_1 in the equation file, then v_1 is taken as the right side expression of the new equation. If there is no truncation value and there are no reduction control options in the procedure call (form a.), the processing will consist of the equation reduction already described. If there is no truncation value but there are reduction control options (form b.), the parameters o_1 and o_2 are the reduction control options that can occur in this form of the procedure call.

An omega option has the form:

OMEGA$ v_1, v_2, ..., v_k

The omega option causes every occurrence of each v_i to be replaced by the variable OMEGA. Then, the identities $\Omega + P = \Omega$ and $\Omega*P = P$ will be applied to the expression prior to expansion.

A phi option has the from:

PHI$ v_1, v_2 ..., v_k

The phi option causes every occurrence of each v_i to be replaced by the variable OMEGA. Then, the identities $\Phi + P = P$ and $\Phi*P = \Phi$ will be applied to the expression prior to expansion. Any number of the o_1 or o_2 options can occur in the procedure call separated by "/" delimiters, and they can occur in any order.

If there is a truncation value but there are no reduction control options (form c.), the expression will be truncated during expansion. The parameter n is a counted literals maximum (ie, the truncation value). Every term which contains more than n variables will be discarded.

If there is a truncation value and there are reduction control options (form d.), the parameters o_1, o_2, o_3, and o_4 are the options for this form of the procedure call. These options may be included in any order. The first option, o_1, is the omega

option and the second option, o_2, is the phi option. These are the same options that were described for form b.

The options o_3 and o_4 are related to n, the counted literals maximum parameter. The option, o_3, is the except complement option and it has one of the following forms:

1. EXCEPTCMP$

2. EXCEPTCMP$ v_1, v_2, ..., v_k

If the except complement option does not have a variable list (form 1.), all complement variables are excluded from counting toward the truncation value. If the except complement option has a variable list (form 2.), only the complement variables corresponding to each v_i in the list are not counted toward the truncation value.

The option, o_4, is the except noncomplement option and it has one of the following forms:

1. EXCEPTNONCMP$

2. EXCEPTNONCMP$ v_1, v_2, ..., v_k

These options function exactly like the except complement options (o_3), but it is the noncomplement variables that are excluded from counting toward the truncation value rather than the complement variables.

Any number of the o_1, o_2, o_3, or o_4 options can occur in the procedure call separated by "/" delimiters, and they can occur in any order.

A.8 Form Block

A call of the Form Block procedure has one of the forms:

a. FRMBLK (b).

b. FRMBLK (b * o_1).

This procedure is used to form a block and add it to the block file. In all forms of the procedure call, the parameter b is the block name for the block to be formed. If there is no selection control option in the procedure call (form a.), a block is formed which contains all of the equations that are in the equation file when the procedure is executed.

If there is a selection control option in the procedure call (form b.), a block will be formed which contains a subset of the equations in the equation file. The selection control option, o_1, will have one of the following forms:

1. ONLY$ v_1, v_2, ..., v_k

2. EXCEPT$ v_1, v_2, ..., v_k

Only one selection control option can occur in a call of the Form Block procedure. If the only option is used (form 1.), the block that is formed will contain only those equations from the equation file that have a left side variable which occurs in the variable list of the option. If the except option is used (form 2.), the block that is formed will contain every equation from the equation file except those that have a left side variable which occurs in the variable list of the option. If there is no equation in the equation file for a variable that occurs in the selection control option, the effect is as if the variable had not occurred in the option.

It is possible to form a block which does not contain any equations, although such a block serves no useful purpose. However, if a selection control option results in excluding all of the equations that are in the equation file, or if there are no equations in the equation file, then a block without any equations will be generated.

A.9 Load Block

A call of the Load Block procedure has the form:

LDBLK (b_1, b_2, ..., b_k).

This procedure is used to load the equations contained in a block into the equation file. The parameters b_1, b_2, ..., b_k are the names of the blocks to be loaded. The parameters are processed from left to right and as each block name is encountered, the equations contained in that block are loaded into the equation file. The blocks in the block file are not affected by this loading process.

If the equation file already contains an equation for some variable v_i, and an equation for v_i is contained in a block to be loaded, the equation for v_i from the block will replace the equation for v_i in the equation file. Otherwise, equations in the equation file will not be changed when a block is loaded. Thus, after each block is loaded, the equation file will consist of all of the equations from the block, together with those equations which were in the equation file when the block was loaded and were not replaced by an equation from the block. Loading a block does not change the block file in any way. Also, if a block is specified for loading which is not in the block file, an error condition will be detected and an error message will be printed.

A block cannot contain more than one equation with the same left side variable because such a block cannot be formed. However, if several blocks are to be loaded, an equation with the same left side variable can occur in more than one of the blocks. Since each block is loaded as its block name is encountered while processing the

parameters b_1, b_2, ..., b_k from left to right, the last equation loaded for a particular variable will be the equation in the equation file when execution of the procedure is completed.

A.10 Print Block

A call of the Print Block procedure has the form:

PRTBLK (b_1, b_2, ..., b_K).

This procedure is used to print the information contained in a block. The parameters b_1, b_2, ..., b_k are the names of the blocks to be printed. As the block names b_1, b_2, ..., b_k are processed from left to right and as each block name is encountered, the information from that block will be printed. If the block was generated by the Read Fault Tree procedure, it contains an internal representation of the fault tree, and the first thing to be printed will be the Fault Tree Event Table. Each event of the fault tree is listed in the Fault Tree Event Table together with the information specifying its relationship to the other events of the fault tree. The numbering of the events begins with 2 because OMEGA is always treated as the first variable in SETS and given the number 1. Since OMEGA cannot occur in a fault tree, it is simply not printed in the Fault Tree Event Table, and the number of events in a fault tree is one less than the number of the last event in the Fault Tree Event Table.

The remainder of the information printed by the Print Block procedure is printed in the same form for all blocks regardless of how they were formed. The equations contained in the block are printed one after the other in the same format used by the Print Equation procedure to print factored equations.

A.11 Delete Block

A call of the Delete Block procedure has one of the forms:

a. DLTBLK.

b. DLTBLK (b_1, b_2, ..., b_k).

This procedure is used to delete blocks from the block file. If there is no parameter list in the procedure call (form a.), all blocks are deleted from the block file. Careful consideration of the consequences should precede the use of this form of the procedure call. However, such a call should occur at the beginning of any SETS user program intended to create a new block file.

If the parameter list is used with the procedure call (form b.), the parameters b_1, b_2, ..., b_k are the names of the blocks to be deleted. Only those blocks with a block name that occurs in the procedure call will be deleted. If more than one block on the block file has the same block name, and if that block name occurs as a parameter in the procedure call, every block with that block name will be deleted from the block file.

A.12 Block Status

A call of the Block Status procedure has the form:

BLKSTAT.

This procedure is used to ascertain what blocks are on the block file. If there are no blocks on the block file, the message

THE BLOCK FILE IS EMPTY

will be printed. If the block file is not empty, the block names of the blocks on the block file will be printed in the same order that they occur on the block file.

APPENDIX B

Execution Diagnostics

During the execution of the SETS program (i.e., during the interpretive execution of a SETS user program), there are several errors that will be detected if they should occur. The errors will be described in two groups. The first group is concerned with the logic and implementation of the SETS program, and the second group concerns errors in a SETS user program.

B.1 SETS Errors

In general, errors detected in the execution of the SETS program indicate a serious breakdown. Although these errors rarely occur, tests are included in SETS to detect them in order to preclude further execution that would produce erroneous results. All of these errors will cause the execution of SETS to be terminated after an appropriate message has been printed.

There are three illegal branch errors that can occur. The messages corresponding to these errors are as follows:

AN ILLEGAL TRANSFER HAS OCCURRED FROM A COMPUTED GOTO STATEMENT.

AN ITERATION PROCESS HAS BEEN COMPLETED WHICH SHOULD HAVE BEEN EXITED PRIOR TO COMPLETION.

THERE HAS BEEN A COMPUTER MALFUNCTION OR AN ERROR EXISTS IN THE SETS PROGRAM.

An illegal branch error will occur if a character is used which is not a valid character in a SETS user program. An illegal branch error may also be caused by a computer malfunction and the job should be run again to make certain that the error was not the result of such a malfunction. An illegal branch error can also occur if a situation occurs that was not anticipated when the SETS program was coded. In this case, the cause of the error must be determined and changes made in the SETS program to correct the error.

There are three file processing errors that can occur. The messages corresponding to these errors are as follows:

AN END OF FILE ERROR HAS OCCURRED.

A PARITY ERROR HAS OCCURRED.

A READY ERROR HAS OCCURRED.

All of the file processing errors can result from a bad file, or from the fact that the file used was not the correct one. The user should first ascertain that the files specified are indeed the ones he intended to use and then run the job again. If the error persists, then the file in question may simply be a bad file and need to be regenerated--particularly if a parity error is occurring. Like the illegal branch errors, file processing errors can be the result of a situation that was not anticipated when the SETS program was coded. A change in the SETS program would then be required to correct the error.

There is one further error that can occur during the execution of the SETS program. This error concerns the printed output generated by SETS, and the message for this error is as follows:

THE MAXIMUM NUMBER OF LINES PER PAGE IS TOO

SMALL TO ALLOW PROPER PAGING OF THE OUTPUT.

This error cannot occur when the standard version of SETS is used. However, the error can occur if a version of SETS is created that reduces the maximum number of printed lines per page to a value that is too small to allow the printing of the headings that can occur in the printed output. This error can be eliminated by using a version of SETS with a larger value for the maximum number of lines per page.

B.2 Sets User Program Errors

The SETS user program errors are syntax errors, and errors that occur during the execution of the statements of the SETS user program. In general, these errors will not cause execution of the SETS program to be terminated. However, the processing of the SETS user program following the detection of one of these errors will be significantly different than normal processing. The processing that occurs after the detection of an error is intended to determine whether or not any remaining input is syntactically correct. It is not possible to accomplish this task completely because recovery after a detected error is based on some syntactic characteristic (eg, the period at the end of each statement). Nevertheless, many of the syntax errors can be detected during a single execution of a SETS user program.

The execution of a SETS user program occurs in two phases. First, the SETS user program itself is read and tested to determine that it is syntactically correct. If an error is detected while reading the SETS user program, an attempt is made to read and test the remainder of the SETS user program unless the error occurred in the program header in which case execution will be terminated. The SETS user program will not be executed if any errors occur while it is being read.

Once the SETS user program can be read without error, its execution will proceed normally unless an error is detected during

execution. If an error is detected during execution, an attempt will be made to execute all remaining procedure calls that process input (RDFT and RDBLK), but execution of all other statements in the SETS user program will be bypassed. However, no blocks will be formed from fault trees or blocks after an error has been detected.

B.2.1 Special Fault Tree Error Messages -- In addition to the numbered error conditions that are described in the next section, there are certain fault tree errors which will cause special messages to be printed. These special messages are as follows:

ERRORS OCCURRED IN THE DEFINITION OF THE EVENT event-name.

THERE WAS NO DEFINITION FOR THE EVENT event-name.

THE DEFINITION FOR THE EVENT event-name DOES NOT INCLUDE ITS RELATIONSHIP TO THE EVENT event-name.

THE RELATIONSHIP BETWEEN THE EVENTS event-name AND event-name IS INCONSISTENT.

where

"event-name" is the name of a fault tree event.

These special messages are the result of tests performed after the entire fault tree has been read. As a result, they provide information which is sometimes already known. For example, when processing the event definition for an event X, if a name is encountered that contains more than the maximum number of name characters allowed, a numbered error message (Error 33) will be printed. Later in the processing the message

ERRORS OCCURRED IN THE DEFINITION OF THE EVENT X

will also be printed even though both messages are the result of the same error. The special messages are helpful in tracking down errors in the fault tree. Correction of the input will then eliminate the errors.

B.2.2 Numbered Error Messages -- Except for the special messages concerning certain fault tree errors, the detection of an error during the execution of a SETS user program will result in a numbered error message of the form:

******ERROR NUMBER: n, name

where

"n" is the error number

"name" is either empty or it is the name of some entity in the SETS user program or its associated input.

The descriptions of the errors that cause numbered error messages are listed below along with the error number that will appear in the message. Furthermore, steps for correcting the error will be indicated if they can be carried out by the user.

Error Number	Error Description
1	A special character is incorrect in the context in which it occurs. The characters that occur in the input between the previous special character and the special character that is incorrect will be printed. Correct the input.
2	A program header, a block header, or a fault tree header is incorrect. The characters that begin the header will be printed. Correct the input.
3	The SETS user program exceeds the size of the vector used to store the program. The name of the SETS user program will be printed. Break up the SETS user program into two or more programs that achieve the same result as the original program.
4	The procedure identifier of a procedure call is incorrect. The incorrect procedure identifier will be printed. Correct the input.
5	The parameter part of a procedure call is incorrect. The procedure identifier of the procedure call will be printed. Correct the input.
6	The block name in a block header, or the fault tree name in a fault tree header is not the same as the next parameter in a RDBLK call or a RDFT call, respectively. The block name or the fault tree name from the header will be printed. Correct the input.
7	The number of variables exceeds the size of the table used to hold them. Initialize the number of variables in the table by inserting a call of DLTEQN with no parameters in the SETS user program after taking steps to save all meaningful equations in blocks. Also, whenever possible, minimize the number of variables in the table before calls of RDFT, RDBLK, and PRTBLK since execution of these procedures temporarily adds additional variables to the table.

8	One of the records of a block exceeds the size of the vector used as a temporary transfer area for a DLTBLK call with a nonempty parameter part. Use a version of SETS with a larger copy vector (equivalenced to the expression vector).
9	The substitution control part of a SUBINEQN call or the reduction control part of a REDUCEQN call is incorrect. Specifically, a variable has occurred more than once in the same kind of control element, or a variable has occurred in both an omega and a phi control element. The variable that caused the error will be printed. Correct the input.
10	The left side variable of an equation is OMEGA. Correct the input.
11	A fault tree does not have any event definitions (i.e., the fault tree body is empty). The fault tree will be printed. Correct the input.
12	A fault tree has an incorrect alphabetic delimiter. The incorrect alphabetic delimiter will be printed. Correct the input.
13	An event definition does not contain any relationship declarations. The name of the event being defined will be printed. Correct the input.
14	A fault tree contains OMEGA as an event name. Correct the input.
15	An event has more than one event definition. The name of the event with multiple definitions will be printed. Correct the input.
16	A fault tree body begins with a relationship declaration instead of an event declaration. The name of the fault tree will be printed. Correct the input.
17	The number of prefixes used in a fault tree exceeds the size of the table used to store them. The prefix that causes the error will be printed. Alter the fault tree so that fewer prefixes are required.

Error Number	Error Description
18	The number of relationships in a fault tree exceeds the size of the vector used to hold them. The fault tree name will be printed. If possible, break up the fault tree into two or more smaller fault trees that represent the same logic as the original fault tree.
19	An event exceeds the maximum rank (i.e., the number of events related to it). The name of the event with the excessive rank will be printed. Insert one or more additional intermediate events into the fault tree so that the logic is preserved, but none of the events in the fault tree exceed the maximum rank.
20	An event in a relationship declaration is the same as the event being defined, or it occurs in more than one relationship declaration in the same event definition. (The same event can occur in a similar input or a similar output declaration if the prefixes are not identical.) The name of the event being defined will be printed. Correct the input.
21	An intermediate event definition with a similar output declaration also has an output declaration or a similar input declaration. The name of the event being defined will be printed. Correct the input.
22	A primary event definition has relationship declarations other than output declarations. The name of the event being defined will be printed. Correct the input.
23	A special intermediate event definition contains a similar input declaration. The name of the event being defined will be printed. Correct the input.
24	An intermediate event definition does not contain any input declarations. The name of the event being defined will be printed. Correct the input.
25	The logic expression in a special intermediate event definition does not contain all of the events that occur in the input declarations. The name of the event being defined will be printed. Correct the input.

Error Number	Error Description
26	The logic expression in a special intermediate event definition contains at least one event that does not occur in an input declaration. The name of the event being defined will be printed. Correct the input.
27	A Boolean expression exceeds the size of the vector used to hold it. Use a version of SETS with a larger expression vector.
28	A conditioning event is related to an event that is not the output event of a PRIORITY AND gate or an INHIBIT gate. The name of the conditioning event will be printed. Correct the input.
29	The output event of a PRIORITY AND gate or an INHIBIT gate does not have exactly one conditioning event related to it. The name of the output event of the PRIORITY AND gate or the INHIBIT gate will be printed. Correct the input.
30	The output event of a PRIORITY AND gate does not have at least two input events related to it. The name of the output event of the PRIORITY AND gate will be printed. Correct the input.
31	The output event of an INHIBIT gate does not have exactly two input events related to it. The name of the output event of the INHIBIT gate will be printed. Correct the input.
32	A fault tree contains at least two similar trees that overlap (i.e., generated event names have more than one prefix). The generated name of the event that causes the error will be printed. Correct the input.
33	The number of characters in a name or alphabetic delimiter exceeds the size of the vector used to build these entities. The first sixteen characters of the name or alphabetic delimiter will be printed. Correct the input.
34	Two special characters (excluding the minus) occur in juxtaposition in a context where such an occurrence is incorrect. Correct the input.

Error Number	Error Description
35	Two special characters (excluding the minus) do not occur in juxtaposition in a context where such an occurrence is required. The characters that occur between the two special characters will be printed. Correct the input.
36	A generated event name is OMEGA, or it is identical to a nongenerated event name, or it is identical to a generated event name but the prefixes are not the same. The generated event name will be printed. Correct the input.
37	The block file does not contain a block with the block name that occurs as a parameter in a LDBLK or a PRTBLK call. The block name will be printed. Correct the input.
38	An equation cannot be printed without exceeding the maximum length allowed for each line of print. Use a version of SETS with a larger maximum printed line length.
39	The right side of an equation is incorrect. Specifically, a variable follows a right parenthesis. The left side variable of the equation will be printed. Correct the input.
40	The right side of an equation is incorrect. Specifically, there is at least one unpaired left parenthesis. The left side variable of the equation will be printed. Correct the input.
41	The right side of an equation is incorrect. Specifically, the period terminating the right side follows the equivalence operator, a left parenthesis, or an operator. The left side variable of the equation will be printed. Correct the input.
42	The right side of an equation is incorrect. Specifically, an AND or OR operator follows a left parenthesis or another operator. The left side variable of the equation will be printed. Correct the input.
43	The right side of an equation is incorrect. Specifically, a NOT operator follows another NOT operator, a right parenthesis, or a variable. The left side variable of the equation will be printed. Correct the input.

Error Number	Error Description
44	The right side of an equation is incorrect. Specifically, a left parenthesis follows a right parenthesis or a variable. The left side variable of the equation will be printed. Correct the input.
45	The right side of an equation is incorrect. Specifically, a right parenthesis follows a left parenthesis or an operator. The left side variable of the equation will be printed. Correct the input.
46	The right side of an equation is incorrect. Specifically, there is at least one unpaired right parenthesis. The left side variable of the equation will be printed. Correct the input.
47	The level of an AND or OR operator exceeds the maximum that can be accommodated during expansion of an expression. The left side variable of the equation which contains the operator will be printed. If possible, break up the equation and reduce it in stages instead of all at once.
48	The left side variable of an equation occurs in the right side of the equation, or the process of substituting equals for equals into the right side of an equation generates a sequence of substitutions that is repetitive and nonending. For the first case the left side variable of the equation will be printed, and for the second case the left side variable from one of the equations in the repetitive sequence will be printed. Correct the input.
49	An integer contains a character other than a digit, or an integer has occurred that is 99999999. Correct the input.
50	The number of variables in a term exceeds the maximum that can be accommodated during the expansion of an expression. Use a counted literals maximum in the REDUCEQN or PRTEQNDNF call to truncate the equation.

Error Number	Error Description
51	The number of equations that either are or have been in the equation file since this numbering was last initialized exceeds the maximum number that can be accommodated. The name of the SETS user program will be printed. Initialize the numbering of equations by inserting a call of DLTEQN with no parameters in the SETS user program after taking steps to save all meaningful equations in blocks.
52	The number of terms in an expression exceeds the size of the vector used to hold them. If this occurs during simplification or factorization of an equation, break up the equation and reduce it in stages instead of all at once, or use a counted literals maximum in the REDUCEQN call to truncate the equation. If it occurs when an equation is being printed in disjunctive normal form, use a counted literals maximum in the PRTEQNDNF call to truncate the equation.
53	The region of Extended Core Storage (ECS) used to store the right sides of equations has been exceeded. The name of the SETS user program will be printed. Initialize the use of ECS by inserting a call of DLTEQN with no parameters in the SETS user program after taking steps to save all meaningful equations in blocks.
54	The number of variables in an expression exceeds the size of the vector used to count the number of occurrences of each literal. If this occurs during factorization, break up the equation and reduce it in stages rather than all at once, or use a counted literals maximum in the REDUCEQN call to truncate the equation. If it occurs when an equation is being printed in disjunctive normal form, use a counted literals maximum in the PRTEQNDNF call to truncate the equation.

Appendix C

The Output of PRTBLK for the Example Sabotage Fault Tree and its Area Equations

* * * * FAULT TREE EVENT TABLE * * * *
(SABOTAGE-FT)

	SET NAME	TYPE	RANK	RELATIONSHIPS
2	TOP	OG	2	IN , FM-TI IN , FM-ILOCA
3	FM-TI	AG	3	IN , TMS-D IN , TI-RT OUT, TOP
4	FM-ILOCA	OG	2	IN , LI-MSD OUT, TOP
5	LI-MSD	AG	3	IN , L-MSD IN , LI OUT, FM-ILOCA
6	L-MSD	OG	4	IN , ECRS-D IN , ECIS-D IN , PAHRS-D OUT, LI-MSD
7	LI	DE	1	OUT, LI-MSD
8	LOSPW	BE	4	OUT, AC-4160-B1J-NP OUT, AC-4160-B1J-AP OUT, AC-4160-B1H-NP OUT, AC-4160-B1H-AP
9	AC-4160-B1J-NP	AG	3	IN , LOSPW IN , AC-4160-B1J-AP OUT, AC-4160-B1J-PS
10	AC-4160-B1J-AP	OG	3	IN , LOSPW IN , COMP-MPWT-D OUT, AC-4160-B1J-NP
11	AC-4160-B1H-NP	AG	3	IN , LOSPW IN , AC-4160-B1H-AP OUT, AC-4160-B1H-PS
12	AC-4160-B1H-AP	OG	3	IN , LOSPW IN , COMP-MPWT-D OUT, AC-4160-B1H-NP
13	RPS-D	DE	1	OUT, TMS-D
14	TMS-D	OG	3	IN , RPS-D IN , DHRS-D OUT, FM-TI

	SET NAME	TYPE	RANK	RELATIONSHIPS
15	ADHRS-D	OG	3	IN , SSRS-D IN , AFWS-LO-IHR OUT, DHRS-D
16	DHRS-D	AG	2	IN , ADHRS-D OUT, TMS-D
17	SSRS-D	DE	1	OUT, ADHRS-D
18	AFWS-LO-IHR	OG	2	IN , AFO-IHR OUT, ADHRS-D
19	TI-RT	OG	3	IN , OI-RC IN , IHR-NHRS OUT, FM-TI
20	OI-RC	DE	1	OUT, TI-RT
21	IHR-NHRS	DE	1	OUT, TI-RT
22	AFO-IHR	OG	4	IN , AFO-PMD-HS IN , AFO-PM-L IN , AFO-CS-PMS OUT, AFWS-LO-IHR
23	AFO-PMD-HS	DE	1	OUT, AFO-IHR
24	AFO-PM-L	AG	3	IN , AFO-MD-P1A IN , AFO-MD-P1B OUT, AFO-IHR
25	AFO-CS-PMS	DE	1	OUT, AFO-IHR
26	AFO-MD-P1A	OG	3	IN , AFO-P1A-D IN , AFO-P1A-EPW OUT, AFO-PM-L
27	AFO-MD-P1B	OG	3	IN , AFO-P1B-D IN , AFO-P1B-EPW OUT, AFO-PM-L
28	AFO-P1A-D	OG	4	IN , AFO-P1A-B IN , AFO-P1A-AUX IN , AFO-P1A-CSG OUT, AFO-MD-P1A
29	AFO-P1A-EPW	OG	2	IN , AC-4160-B1H OUT, AFO-MD-P1A
30	AFO-P1A-B	BE	1	OUT, AFO-P1A-D
31	AFO-P1A-AUX	OG	2	IN , AFO-P1A-COOL

	SET NAME	TYPE	RANK	RELATIONSHIPS
				OUT, AFO-P1A-D
32	AFO-P1A-CSG	BE	1	OUT, AFO-P1A-D
33	AC-4160-B1H	OG	6	IN , EP-BS-1H-D IN , AC-4160-B1H-PS OUT, AFO-P1A-EPW OUT, SWA-PM-L OUT, CHO-P1A-EPW OUT, AC-480V-B480H-PS
34	AFO-P1A-COOL	DE	1	OUT, AFO-P1A-AUX
35	AFO-P1B-D	OG	4	IN , AFO-P1B-B IN , AFO-P1B-AUX IN , AFO-P1B-CSG OUT, AFO-MD-P1B
36	AFO-P1B-EPW	OG	2	IN , AC-4160-B1J OUT, AFO-MD-P1B
37	AFO-P1B-B	BE	1	OUT, AFO-P1B-D
38	AFO-P1B-AUX	OG	2	IN , AFO-P1B-COOL OUT, AFO-P1B-D
39	AFO-P1B-CSG	BE	1	OUT, AFO-P1B-D
40	AC-4160-B1J	OG	6	IN , EP-BS-1J-D IN , AC-4160-B1J-PS OUT, AFO-P1B-EPW OUT, SWB-PM-L OUT, CHO-P1B-EPW OUT, AC-480V-B480J-PS
41	AFO-P1B-COOL	DE	1	OUT, AFO-P1B-AUX
42	ECRS-D	AG	3	IN , CS1A-IHR IN , CS1B-IHR OUT, L-MSD
43	ECIS-D	OG	2	IN , CHO-IHR OUT, L-MSD
44	PAHRS-D	UE	1	OUT, L-MSD
45	CS1A-IHR	OG	5	IN , CS1A-PMD-HS IN , CS1A-HS IN , CS1A-PM-L IN , CS1A-HS-PMS OUT, ECRS-D

	SET NAME	TYPE	RANK	RELATIONSHIPS
46	CS1B-IHR	OG	5	IN , CS1B-PMD-HS IN , CS1B-HS IN , CS1B-PM-L IN , CS1B-HS-PMS OUT, ECRS-D
47	CHO-IHR	OG	4	IN , CHO-PMD-HS IN , CHO-CS-PMS IN , CHO-PM-L OUT, ECIS-D
48	CS1A-PMD-HS	OG	3	IN , CS1A-CX-SUMP-B IN , CS1A-PP-PMD-B OUT, CS1A-IHR
49	CS1A-HS	DE	1	OUT, CS1A-IHR
50	CS1A-PM-L	^G	2	IN , CS1A-MD-P1A OUT, CS1A-IHR
51	CS1A-HS-PMS	OG	2	IN , CS1A-PP-PMS-B OUT, CS1A-IHR
52	CS1A-CX-SUMP-B	BE	1	OUT, CS1A-PMD-HS
53	CS1A-PP-PMD-B	BE	1	OUT, CS1A-PMD-HS
54	CS1A-MD-P1A	OG	3	IN , CS1A-P1A-D IN , CS1A-P1A-EPW OUT, CS1A-PM-L
55	CS1A-PP-PMS-B	BE	1	OUT, CS1A-HS-PMS
56	CS1A-P1A-D	OG	3	IN , CS1A-P1A-B IN , CS1A-P1A-CSG OUT, CS1A-MD-P1A
57	CS1A-P1A-EPW	OG	2	IN , AC-480V-B480H OUT, CS1A-MD-P1A
58	CS1A-P1A-B	BE	1	OUT, CS1A-P1A-D
59	CS1A-P1A-CSG	BE	1	OUT, CS1A-P1A-D
60	AC-480V-B480H	OG	3	IN , AC-480V-B480H-PS IN , ID-BS-480H-D OUT, CS1A-P1A-EPW
61	CS1B-PMD-HS	OG	3	IN , CS1B-CX-SUMP-B IN , CS1B-PP-PMD-B OUT, CS1B-IHR

	SET NAME	TYPE	RANK	RELATIONSHIPS
62	CS1B-HS	DE	1	OUT, CS1B-IHR
63	CS1B-PM-L	AG	2	IN , CS1B-MD-P1B OUT, CS1B-IHR
64	CS1B-HS-PMS	OG	2	IN , CS1B-PP-PMS-B OUT, CS1B-IHR
65	CS1B-CX-SUMP-B	BE	1	OUT, CS1B-PMD-HS
66	CS1B-PP-PMD-B	BE	1	OUT, CS1B-PMD-HS
67	CS1B-MD-P1B	OG	3	IN , CS1B-P1B-D IN , CS1B-P1B-EPW OUT, CS1B-PM-L
68	CS1B-PP-PMS-B	BE	1	OUT, CS1B-HS-PMS
69	CS1B-P1B-D	OG	3	IN , CS1B-P1B-B IN , CS1B-P1B-CSG OUT, CS1B-MD-P1B
70	CS1B-P1B-EPW	OG	2	IN , AC-480V-B480J OUT, CS1B-MD-P1B
71	CS1B-P1B-B	BE	1	OUT, CS1B-P1B-D
72	CS1B-P1B-CSG	BE	1	OUT, CS1B-P1B-D
73	AC-480V-B480J	OG	3	IN , AC-480V-B480J-PS IN , ID-BS-480J-D OUT, CS1B-P1B-EPW
74	SWA-IHR	OG	5	IN , SWA-PMD-HS IN , SWA-CS-PMS IN , SWA-PM-L OUT, CHO-P1A-COOL OUT, CHO-P1B-COOL
75	SWA-PMD-HS	OG	4	IN , SWA-HX-OILCO-B IN , SWA-PP-PMD-B IN , SWA-VV-PMD OUT, SWA-IHR
76	SWA-CS-PMS	DE	1	OUT, SWA-IHR
77	SWA-PM-L	OG	3	IN , AC-4160-B1H IN , SWA-MD-P10A OUT, SWA-IHR
78	CHO-P1A-COOL	AG	3	IN , SWA-IHR IN , SWB-IHR

	SET NAME	TYPE	RANK	RELATIONSHIPS
				OUT, CHO-P1A-AUX
79	CHO-P1B-COOL	AG	3	IN , SWA-IHR IN , SWB-IHR OUT, CHO-P1B-AUX
80	SWA-HX-OILCO-B	BE	1	OUT, SWA-PMD-HS
81	SWA-PP-PMD-B	BE	1	OUT, SWA-PMD-HS
82	SWA-VV-PMD	BE	1	OUT, SWA-PMD-HS
83	SWA-MD-P10A	DE	1	OUT, SWA-PM-L
84	SWB-IHR	OG	5	IN , SWB-PMD-HS IN , SWB-CS-PMS IN , SWB-PM-L OUT, CHO-P1A-COOL OUT, CHO-P1B-COOL
85	SWB-PMD-HS	OG	4	IN , SWB-HX-OILCO-B IN , SWB-PP-PMD-B IN , SWB-VV-PMD OUT, SWB-IHR
86	SWB-CS-PMS	DE	1	OUT, SWB-IHR
87	SWB-PM-L	OG	3	IN , AC-4160-B1J IN , SWB-MD-P10B OUT, SWB-IHR
88	SWB-HX-OILCO-B	BE	1	OUT, SWB-PMD-HS
89	SWB-PP-PMD-B	BE	1	OUT, SWB-PMD-HS
90	SWB-VV-PMD	BE	1	OUT, SWB-PMD-HS
91	SWB-MD-P10B	DE	1	OUT, SWB-PM-L
92	CHO-PMD-HS	OG	4	IN , CHO-HX-RVESS-B IN , CHO-VV-PMD IN , CHO-PP-PMD-B OUT, CHO-IHR
93	CHO-CS-PMS	DE	1	OUT, CHO-IHR
94	CHO-PM-L	AG	3	IN , CHO-MD-P1A IN , CHO-MD-P1B OUT, CHO-IHR
95	CHO-HX-RVESS-B	BE	1	OUT, CHO-PMD-HS

	SET NAME	TYPE	RANK	RELATIONSHIPS
96	CHO-VV-PMD	OG	4	IN , CHO-MV1286ABCOC IN , CHO-MV1867ABCC IN , CHO-MV1867BOCC OUT, CHO-PMD-HS
97	CHO-PP-PMD-B	BE	1	OUT, CHO-PMD-HS
98	CHO-MD-P1A	OG	3	IN , CHO-P1A-D IN , CHO-P1A-EPW OUT, CHO-PM-L
99	CHO-MD-P1B	OG	3	IN , CHO-P1B-D IN , CHO-P1B-EPW OUT, CHO-PM-L
100	CHO-MV1286ABCOC	BE	1	OUT, CHO-VV-PMD
101	CHO-MV1867ABCC	BE	1	OUT, CHO-VV-PMD
102	CHO-MV1867BOCC	BE	1	OUT, CHO-VV-PMD
103	CHO-P1A-D	OG	4	IN , CHO-P1A-B IN , CHO-P1A-AUX IN , CHO-P1A-CSG OUT, CHO-MD-P1A
104	CHO-P1A-EPW	OG	2	IN , AC-4160-B1H OUT, CHO-MD-P1A
105	CHO-P1A-B	BE	1	OUT, CHO-P1A-D
106	CHO-P1A-AUX	OG	2	IN , CHO-P1A-COOL OUT, CHO-P1A-D
107	CHO-P1A-CSG	BE	1	OUT, CHO-P1A-D
108	CHO-P1B-D	OG	4	IN , CHO-P1B-B IN , CHO-P1B-AUX IN , CHO-P1B-CSG OUT, CHO-MD-P1B
109	CHO-P1B-EPW	OG	2	IN , AC-4160-B1J OUT, CHO-MD-P1B
110	CHO-P1B-B	BE	1	OUT, CHO-P1B-D
111	CHO-P1B-AUX	OG	2	IN , CHO-P1B-COOL OUT, CHO-P1B-D
112	CHO-P1B-CSG	BE	1	OUT, CHO-P1B-D
113	EP-BS-1J-D	BE	1	OUT, AC-4160-B1J

	SET NAME	TYPE	RANK	RELATIONSHIPS
114	AC-4160-B1J-PS	AG	3	IN , AC-4160-B1J-NP IN , AC-4160-B1J-SB OUT, AC-4160-B1J
115	AC-480V-B480J-PS	OG	3	IN , AC-4160-B1J IN , ID-TR-SSXFM1J-D OUT, AC-480V-B480J
116	AC-4160-B1J-SB	OG	3	IN , CB-C1J-O IN , DG-NO3-L OUT, AC-4160-B1J-PS
117	CB-C1J-O	BE	1	OUT, AC-4160-B1J-SB
118	DG-NO3-L	OG	4	IN , DG-NO3-CSG IN , DG-NO3-B IN , DG-NO3-AUX OUT, AC-4160-B1J-SB
119	COMP-MPWT-D	UE	2	OUT, AC-4160-B1J-AP OUT, AC-4160-B1H-AP
120	EP-BS-1H-D	BE	1	OUT, AC-4160-B1H
121	AC-4160-B1H-PS	AG	3	IN , AC-4160-B1H-NP IN , AC-4160-B1H-SB OUT, AC-4160-B1H
122	AC-480V-B480H-PS	OG	3	IN , AC-4160-B1H IN , ID-TR-SSXFM1H-D OUT, AC-480V-B480H
123	AC-4160-B1H-SB	OG	3	IN , CB-C1H-O IN , DG-NO1-L OUT, AC-4160-B1H-PS
124	CB-C1H-O	BE	1	OUT, AC-4160-B1H-SB
125	DG-NO1-L	OG	4	IN , DG-NO1-CSG IN , DG-NO1-B IN , DG-NO1-AUX OUT, AC-4160-B1H-SB
126	DG-NO3-CSG	DE	1	OUT, DG-NO3-L
127	DG-NO3-B	BE	1	OUT, DG-NO3-L
128	DG-NO3-AUX	DE	1	OUT, DG-NO3-L
129	DG-NO1-CSG	DE	1	OUT, DG-NO1-L
130	DG-NO1-B	BE	1	OUT, DG-NO1-L

	SET NAME	TYPE	RANK	RELATIONSHIPS
131	DG-NO1-AUX	DE	1	OUT, DG-NO1-L
132	ID-BS-480H-D	BE	1	OUT, AC-480V-B480H
133	ID-TR-SSXFM1H-D	BE	1	OUT, AC-480V-B480H-PS
134	ID-BS-480J-D	BE	1	OUT, AC-480V-B480J
135	ID-TR-SSXFM1J-D	BE	1	OUT, AC-480V-B480J-PS

TOP = FM-TI + FM-ILOCA

FM-TI = TMS-D * TI-RT

FM-ILOCA = LI-MSD

LI-MSD = L-MSD * LI

L-MSD = ECRS-D + ECIS-D + PAHRS-D

AC-4160-B1J-NP = LOSPW * AC-4160-B1J-AP

AC-4160-B1J-AP = LOSPW + COMP-MPWT-D

AC-4160-B1H-NP = LOSPW * AC-4160-B1H-AP

AC-4160-B1H-AP = LOSPW + COMP-MPWT-D

TMS-D = RPS-D + DHRS-D

ADHRS-D = SSRS-D + AFWS-LO-IHR

DHRS-D = ADHRS-D

AFWS-LO-IHR = AFO-IHR

TI-RT = OI-RC + IHR-NHRS

AFO-IHR = AFO-PMD-HS + AFO-PM-L + AFO-CS-PMS

AFO-PM-L = AFO-MD-P1A * AFO-MD-P1B

AFO-MD-P1A = AFO-P1A-D + AFO-P1A-EPW

AFO-MD-P1B = AFO-P1B-D + AFO-P1B-EPW

AFO-P1A-D = AFO-P1A-B + AFO-P1A-AUX + AFO-P1A-CSG

AFO-P1A-EPW = AC-4160-B1H

AFO-P1A-AUX = AFO-P1A-COOL

AC-4160-B1H = EP-BS-1H-D + AC-4160-B1H-PS

AFO-P1B-D = AFO-P1B-B + AFO-P1B-AUX + AFO-P1B-CSG

AFO-P1B-EPW = AC-4160-B1J

AFO-P1B-AUX = AFO-P1B-COOL

AC-4160-B1J = EP-BS-1J-D + AC-4160-B1J-PS

ECRS-D = CS1A-IHR * CS1B-IHR

ECIS-D = CHO-IHR

CS1A-IHR = CS1A-PMD-HS + CS1A-HS + CS1A-PM-L +

CS1A-HS-PMS

CS1B-IHR = CS1B-PMD-HS + CS1B-HS + CS1B-PM-L +

CS1B-HS-PMS

CHO-IHR = CHO-PMD-HS + CHO-CS-PMS + CHO-PM-L

CS1A-PMD-HS = CS1A-CX-SUMP-B + CS1A-PP-PMD-B

CS1A-PM-L = CS1A-MD-P1A

CS1A-HS-PMS = CS1A-PP-PMS-B

CS1A-MD-P1A = CS1A-P1A-D + CS1A-P1A-EPW

CS1A-P1A-D = CS1A-P1A-B + CS1A-P1A-CSG

CS1A-P1A-EPW = AC-480V-B480H

AC-480V-B480H = AC-480V-B480H-PS + ID-BS-480H-D

CS1B-PMD-HS = CS1B-CX-SUMP-B + CS1B-PP-PMD-B

CS1B-PM-L = CS1B-MD-P1B

CS1B-HS-PMS = CS1B-PP-PMS-B

CS1B-MD-P1B = CS1B-P1B-D + CS1B-P1B-EPW

CS1B-P1B-D = CS1B-P1B-B + CS1B-P1B-CSG

CS1B-P1B-EPW = AC-480V-B480J

AC-480V-B480J = AC-480V-B480J-PS + ID-BS-480J-D

SWA-IHR = SWA-PMD-HS + SWA-CS-PMS + SWA-PM-L

SWA-PMD-HS = SWA-HX-OILCO-B + SWA-PP-PMD-B + SWA-VV-PMD

SWA-PM-L = AC-4160-B1H + SWA-MD-P10A

CHO-P1A-COOL = SWA-IHR * SWB-IHR

CHO-P1B-COOL = SWA-IHR * SWB-IHR

SWB-IHR = SWB-PMD-HS + SWB-CS-PMS + SWB-PM-L

SWB-PMD-HS = SWB-HX-OILCO-B + SWB-PP-PMD-B + SWB-VV-PMD

SWB-PM-L = AC-4160-B1J + SWB-MD-P10B

CHO-PMD-HS = CHO-HX-RVESS-B + CHO-VV-PMD + CHO-PP-PMD-B

CHO-PM-L = CHO-MD-P1A * CHO-MD-P1B

CHO-VV-PMD = CHO-MV1286ABCOC + CHO-MV1867ABCC +

CHO-MV1867BDCC

CHO-MD-P1A = CHO-P1A-D + CHO-P1A-EPW

CHO-MD-P1B = CHO-P1B-D + CHO-P1B-EPW

CHO-P1A-D = CHO-P1A-B + CHO-P1A-AUX + CHO-P1A-CSG

CHO-P1A-EPW = AC-4160-B1H

CHO-P1A-AUX = CHO-P1A-COOL

CHO-P1B-D = CHO-P1B-B + CHO-P1B-AUX + CHO-P1B-CSG

CHO-P1B-EPW = AC-4160-B1J

CHO-P1B-AUX = CHO-P1B-COOL

AC-4160-B1J-PS = AC-4160-B1J-NP * AC-4160-B1J-SB

AC-480V-B480J-PS = AC-4160-B1J + ID-TR-SSXFM1J-D

AC-4160-B1J-SB = CB-C1J-O + DG-NO3-L

DG-NO3-L = DG-NO3-CSG + DG-NO3-B + DG-NO3-AUX

AC-4160-B1H-PS = AC-4160-B1H-NP * AC-4160-B1H-SB

AC-480V-B480H-PS = AC-4160-B1H + ID-TR-SSXFM1H-D

AC-4160-B1H-SB = CB-C1H-O + DG-NO1-L

DG-NO1-L = DG-NO1-CSG + DG-NO1-B + DG-NO1-AUX

IHR-NHRS = SFGRDB

RPS-D = SFGRDA

OI-RC = SFGRDB

SSRS-D = MCC1JCV + MCC1HCV

AFO-PMD-HS = CR + MCC1JCV

AFO-CS-PMS = SFGRDA

AFO-P1A-B = MCC1HCV

AFO-P1A-CSG = MCC1HCV

AFO-P1A-COOL = DGRM2

AFO-P1B-B = MCC1JCV

AFO-P1B-CSG = MCC1JCV

AFO-P1B-COOL = DGRM1 + ESGRM

EP-BS-1J-D = ESGRM

CB-C1J-O = ESGRM

EP-BS-1H-O = ESGRM

CB-C1H-O = ESGRM

DG-NO3-B = DGRM2

DG-NO1-B = DGRM1

ID-BS-480H-D = WSGRM

ID-TR-SSXFM1H-D = 2AB

ID-BS-480J-D = ESGRM

ID-TR-SSXFM1J-D = 3EQUIPRM

AFWS-LO-IHR = CR + MCC1HCV * MCC1JCV + SFGRDA

CS1A-PP-PMD-B = 3EQUIPRM

CS1A-CX-SUMP-B = CR

CS1A-PP-PMS-B = CR + 2AB

CS1A-P1A-B = PMPHSE

CS1A-P1A-CSG = CR

CS1B-PP-PMD-B = CR

CS1B-CX-SUMP-B = PMPHSE

CS1B-PP-PMS-B = CR + 2AB * 2RB

CS1B-P1B-B = CR

CS1B-P1B-CSG = CR

SWA-HX-OILCO-B = TBSMTVVPT

SWA-PP-PMD-B = 3EQUIPRM + ESGRM * WSGRM + 2AB

SWA-VV-PMD = 3EQUIPRM + 2AB

SWB-HX-OILCO-B = CHPMCUB

SWB-PP-PMD-B = 3EQUIPRM + ESGRM * WSGRM + 2AB

SWB-VV-PMD = 3EQUIPRM + 2AB

CHO-HX-RVESS-B = CR

CHO-PP-PMD-B = CHPMCUB + 2AB

CHO-MV1286ABCOC = CR + MCC1HCV * MCC1JCV + CHPMCUB

CHO-MV1867ABCC = CR + MCC1HCV * MCC1JCV + 2AB

CHO-MV1867BDCC = CR + MCC1JCV * MCC1HCV + 2AB

CHO-P1A-B = CHPMCUB

CHO-P1A-CSG = CHPMCUB

CHO-P1B-B = ESGRM

CHO-P1B-CSG = CHPMCUB

LI = CHPMCUB + 2AB + ESGRM + CR + RC

CS1A-HS = SFGRDA + MCC1HCV

CS1B-HS = SFGRDA + MCC1JCV + WSGRM

SWA-CS-PMS = TBSMTVVPT

SWB-CS-PMS = TBSMTVVPT

SWA-MD-P10A = CR + 3EQUIPRM

SWB-MD-P10B = CR + 3EQUIPRM

CHO-CS-PMS = RWST + 2RWST + CR + 2AB + CHPMCUB

```
DG-NO3-AUX = DGRM2

DG-NO1-AUX = DGRM1

DG-NO3-CSG = CR + RLYRM + DGRM1 * DGRM2

DG-NO1-CSG = CR + RLYRM + DGRM1 * DGRM2
```

Distribution:

US NRC Distribution Contractor (CDSI)
7300 Pearl Street
Bethesda, MD 20014
230 copies for RS

US Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Washington, DC 20555
Attn: W. C. Floyd/DFQ (5)
J. A. Murphy/DRA
D. M. Rasmuson/DRA

US Nuclear Regulatory Commission
Office of Nuclear Material Safety and Safeguards
Washington, DC 20555
Attn: R. F. Burnett/SG (2)
G. W. McCorkle/SG

US Nuclear Regulatory Commission
Office of Nuclear Reactor Regulation
Washington, DC 20555
Attn: R. J. Mattson/DSI

Science Applications, Inc.
P. O. Box 2351
La Jolla, CA 92038
Attn: P. Lobner (10)

International Energy Associates, Ltd.
1126 Santa Ana SE
Albuquerque, NM 87123
Attn: G. B. Varnado

Science and Engineering Associates
2500 Louisiana Boulevard NE
Suite 610
Albuquerque, NM 87110
Attn: J. L. Darby

University of California
School of Engineering and Applied Sciences
5532 Boelter Hall
Los Angeles, CA 90024
Attn: David Okrent

Energy Incorporated
1851 South Central Place
Suite 201
Kent, WA 98031
Attn: Jon Young

Distribution (Cont.)

GA Technologies
P. O. Box 81608
San Diego, CA 92138
Attn: M. G. Stamatelatos

Los Alamos National Laboratory
Mail Stop: G777
Los Alamos, NM 87545
Attn: R. Haarman

3141-1	C. M. Ostrander (5)
3151	W. L. Garner
6400	A. W. Snyder
6410	J. W. Hickman
6411	D. M. Kunsman
6412	F. T. Harper
6412	S. W. Hatch
6412	G. J. Kolb
6412	A. C. Payne, Jr.
6412	D. W. Stack (16)
6412	T. A. Wheeler
6412	D. W. Whitehead
6414	D. M. Ericson
6414	W. R. Cramond
6414	S. L. Daniel
6414	D. R. Gallup
6414	G. A. Sanders
6415	D. C. Aldrich
6417	D. D. Carlson
6430	N. R. Ortiz
6432	L. D. Chapman
6432	R. B. Worrell
6447	W. T. Wheelis
8424	M. A. Pound

NRC FORM 335 (11.81)

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

1. REPORT NUMBER (Assigned by DDC)
NUREG/CR-3134
SAND83-0074

4. TITLE AND SUBTITLE (Add Volume No., if appropriate)

A SETS USER'S MANUAL FOR VITAL AREA ANALYSIS

2. (Leave blank)

3. RECIPIENT'S ACCESSION NO.

7. AUTHOR(S)
Desmond W. Stack
Mildred S. Hill

5. DATE REPORT COMPLETED
MONTH: March YEAR: 1984

9. PERFORMING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)

Sandia National Laboratories
Albuquerque, New Mexico 87185

DATE REPORT ISSUED
MONTH: March YEAR: 1984

6. (Leave blank)

8. (Leave blank)

12. SPONSORING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)

Division of Facility Operations
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555

10. PROJECT/TASK/WORK UNIT NO.

11. FIN NO.
NRC FIN No. A1338

13. TYPE OF REPORT
Technical Report

PERIOD COVERED (Inclusive dates)

15. SUPPLEMENTARY NOTES

14. (Leave blank)

16. ABSTRACT (200 words or less)

This manual describes the use of the Set Equation Transformation System (SETS) for vital area analysis. Various techniques are presented for using SETS to solve vital area analysis fault trees. Depending on the input to SETS, the solution to the vital area analysis fault tree can be in terms of vital areas or primary events of the vital area analysis fault tree. The techniques presented are also suitable and efficient for other kinds of common cause analysis.

17. KEY WORDS AND DOCUMENT ANALYSIS

17a. DESCRIPTORS

Fault Trees
Minimal Cut Set Equations
Common Cause Analysis
Variable Transformations

17b. IDENTIFIERS/OPEN-ENDED TERMS

18. AVAILABILITY STATEMENT

Unlimited

19. SECURITY CLASS (This report)
Unclassified

21. NO. OF PAGES

20. SECURITY CLASS (This page)
Unclassified

22. PRICE
$